HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr Ing.
Marvin Seyfarth

Konzept für Testautomatisierung am Beispiel des Testobjekts robotron*ecount

Mittweida, 2017

Fakultät Angewandte Computer- und Biowissenschaften

DIPLOMARBEIT

Konzept für Testautomatisierung am Beispiel des Testobjekts robotron*ecount

Autor:
Herr Ing. Marvin Seyfarth

Studiengang:
Informationstechnik

Seminargruppe:
KI13w1-F

Erstprüfer:
Prof. Dr. Ing. Wilfried Schubert

Zweitprüfer:
Tech. Dipl.-Betriebswirt (FH) Christian Kunert

Einreichung:
Mittweida, 11.10.2017

Verteidigung/Bewertung:
Mittweida, 2017

Bibliografische Beschreibung:

Seyfarth, Marvin:
Konzept für Testautomatisierung am Beispiel des Testobjekts robotron*ecount. - 2017. - 9, 48, 14 S.
Mittweida, Hochschule Mittweida, Fakultät Angewandte Computer- und Biowissenschaften, Diplomarbeit, 2017

Referat:

In der vorliegenden Arbeit soll ein Lösungskonzept für Regressionstests des Softwareprodukts robotron*ecount entstehen. Ziel ist die Ermöglichung eines effizienten und wirtschaftlichen Regressionstests mithilfe der Testautomatisierung. Zunächst wird ein universeller Testprozess definiert. Des Weiteren erfolgt eine Untersuchung von technischen und logischen Konzepten sowie Werkzeugen bezüglich der Testautomatisierung wie z. B. Formen der Testfalldarstellung und Testmetriken. Außerdem wird die Struktur und Arbeitsweise des Testobjekts robotron*ecount analysiert. Die gewonnenen Erkenntnisse werden anschließend bei der Gestaltung des Lösungskonzepts angewendet.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

API	Application programming interface
EDI	Electronic Data Interchange
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport
ERM	Entity-Relationship-Modell
GeLi Gas	Geschäftsprozesse Lieferantenwechsel Gas
GPKE	Geschäftsprozesse zur Kundenbelieferung mit Elektrizität
GUI	Graphical User Interface
IT	Informationstechnik
KNN	Künstliche neuronale Netze
LFA	Lieferant Alt
LFN	Lieferant Neu
MPES	Marktprozesse für Erzeugungsanlagen (Strom)
NB	Netzbetreiber
TK	Telekommunikation
UTILMD	Utilities Master Data message
WiM	Wechselprozesse im Messwesen
XML	Extensible Markup Language

1 Einleitung und Motivation

Die Mitteldeutsche Netzgesellschaft mbH (MITNETZ) ist als größter regionaler Verteilnetzbetreiber in Ostdeutschland für Planung, Betrieb und Vermarktung des Strom- und Gasnetzes verantwortlich. Eines der Hauptziele ist die Umsetzung von regulatorischen Vorgaben sowie eine diskriminierungsfreie Ausübung des Strom- und Gasnetzbetriebes. Die MITNETZ betreibt ein Netzgebiet für Strom und Gas von über 35.000 km² über Teile der Bundesländer Brandenburg, Sachsen, Sachsen-Anhalt und Thüringen.

„Das Aufgabenspektrum meiner Organisationseinheit Netzzugangsmanagement umfasst die Abwicklung von Geschäftsprozessen zur Kundenbelieferung mit Elektrizität (GPKE), Lieferantenwechsel Gas (GeLi Gas), Messwesen (WiM) und Erzeugungsanlagen (MPES). Die Umsetzung der Geschäftsprozesse erfolgt mithilfe der JAVA-basierten robotron*ecount-Software sowie Oracle als Datenbanksystem. Die Bereitstellung und Wartung dieser Software wird durch den externen Dienstleister Robotron Datenbank-Software GmbH (robotron) sichergestellt. Aus den Beschlüssen der Bundesnetzagentur resultieren kontinuierliche Anpassungen der Geschäftsprozesse und der EDIFACT-Formatbeschreibungen. Die Softwarelösung unterliegt somit ständigen Änderungen. Demzufolge besteht die Anforderung, Anpassungen der Software unter bestimmten Rahmenbedingungen auf Richtigkeit zu überprüfen.“ [1] S. 1

Im Modul „Praxisprojekt“ des Diplomstudiengangs Informationstechnik (Fernstudiengang) wurden geeignete Softwarelösungen zur Unterstützung eines Regressionstests ausgewählt, bewertet und eingeordnet. Aus den Erkenntnissen der daraus entstandenen Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ ist die Anforderung entstanden, dass für eine erfolgreiche und wirtschaftliche Ausführung eines Regressionstests eine weitestgehende Automation der Testvorbereitung, Testdurchführung und Testdokumentation erfolgen muss.

In dieser Arbeit sollen das Testobjekt robotron*ecount sowie ausgewählte Testtools auf die Eigenschaft zur Unterstützung zur Testautomatisierung hin untersucht werden. In einem Lösungskonzept wird die Verwendung und Zusammenarbeit der vorhandenen Testtools arrangiert und um neue Lösungen ergänzt.

1.1 Ist-Zustand

„Momentan muss jede Auslieferung von robotron getestet werden, um Fehler in der Software auszuschließen. Fehler führen zu Aufwand und Kosten bezüglich Analyse und Behebung. Des Weiteren können Softwarefehler die Einhaltung der von der Bundesnetzagentur

vorgegebenen Geschäftsprozesse verzögern oder verhindern und damit zu Kundenbeschwerden und behördlichen Geldstrafen führen. Momentan werden nur die ausgelieferten Funktionen getestet, oft treten aber an anderen unangepassten Funktionen Fehler auf. Mit den aktuell vorhandenen Mitteln ist eine vollumfängliche Überprüfung der Software nicht möglich. […]“ [1] S. 1

Aus der Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ ist die Anforderung eines Regressionstest entstanden [1] S. 16. Diese Anforderung konnte bisher nicht erfüllt werden. Die in der gleichen Arbeit untersuchten Testtools ermöglichen weiterhin keine ausreichende Testautomatisierung. Aktuell bestehen keine bewusst einheitliche Verwendung der Testtools sowie kein Konzept zur Erhöhung des Testautomatisierungsgrades. Die Testautomatisierung ist aber eine wesentliche Grundlage für wirtschaftliche und erfolgreiche Regressionstests.

1.2 Soll-Zustand

Zunächst soll eine Abgrenzung und Einordnung von Begriffen, Konzepten und Werkzeugen zum Thema Testautomatisierung erfolgen. Bezüglich dieser Grundlage wird die Arbeit dann anschließend gegliedert. Zusammen mit den Kenntnissen bezüglich der Struktur des Testobjekts und der vorhandenen Testtools soll zunächst ein Regressionstest definiert werden. Darauf aufbauend soll ein Lösungskonzept zur Erhöhung des Automationsgrades entstehen. Ziel ist das ermöglichen eines effizienten und wirtschaftlichen Regressionstests mithilfe der Testautomatisierung.

Die in der Anlage Teil 1 hinterlegte Aspektübersicht zum Begriff Testautomatisierung bildet den strukturellen Rahmen zur wissenschaftlichen Untersuchung des Themas. Diese Struktur nimmt auf die Gliederung der Literatur „Basiswissen Testautomatisierung“ [2] Bezug und wurde weiter modifiziert und ergänzt.

Neben der weitreichenden technischen Auseinandersetzung mit dem Thema Testautomatisierung soll auch eine kurze Behandlung der Auswirkungen der Testautomatisierung auf den Arbeitsprozess erfolgen. Eine vollumfängliche Untersuchung diesbezüglich würde über den Rahmen dieser Arbeit hinausgehen. Des Weiteren wird auch das Thema „Softwarequalität“ trotz der Nähe zur Testautomatisierung nicht näher behandelt. Die Implementierung und Integration des erstellten Lösungskonzepts soll im Rahmen dieser Arbeit nicht stattfinden.

2 Testautomatisierung

Auf den folgenden Seiten werden die Begriffe „Testautomatisierung“ und „Testprozess“ näher definiert sowie technische und logische Konzepte der Testautomatisierung untersucht. Außerdem erfolgt eine allgemeine Aspektanalyse von Testwerkzeugen.

2.1 Terminologie Testautomatisierung

Die Grundlage jeglicher Automatisierung sind Algorithmen bzw. Heuristiken. Ziel ist es, durch entsprechend hohen Initialaufwand eine wiederkehrende Aufgabe mit geringen Ressourcen (Zeit, Arbeitskraft) zu bewältigen. Übersteigt die Ressourceneinsparung im laufenden Testbetrieb den Ressourcenaufwand in der Vorbereitung der Automation, dann ist die Testautomatisierung als wirtschaftlich erfolgreich anzusehen. Des Weiteren kann durch Testautomatisierung die Testfallanzahl effektiv erhöht werden. Dies ermöglicht die Erhebung von aussagekräftigen Kennzahlen, deren Interpretation und Auswertung die Testqualität und damit die Softwarequalität verbessern.

2.2 Chancen und Risiken der Testautomatisierung

In diesem Abschnitt wird geklärt, ob eine Testautomatisierung für Regressionstests möglich und auch sinnvoll ist. Bei jeder Änderung am System müssen schließlich auch Testfallparameter im Regressionstest geändert werden.

Voraussetzung für eine effektive Testautomatisierung sind standardisierte Prozesse und Datenmodelle des Testobjekts. Ein weiterer großer Umstand für die Automatisierung des Testprozesses ist die fortlaufende Änderung der zu testenden Software (Testobjekt). Anpassungen können ganze - aufwendig erstellte - Automatisierungspläne oder Teile davon verwerfen. Deswegen müssen Änderungen am Testobjekt strukturiert dokumentiert werden, damit eine schnelle Anpassung der automatischen Testfälle erfolgen kann. Ist die Automatisierung erst mal erfolgreich eingestellt, lassen sich erhebliche Personalressourcen einsparen. Auch wenn bereits im Vorfeld wenig Testpersonal abgestellt ist, lässt sich die Quantität der Testfälle erheblich erhöhen. Durch die erleichterte Testfallerstellung und Testdurchführung wird dem Testpersonal viel Aufwand und dadurch viel Zeit erspart, z. B. durch das Automatisieren von wiederkehrenden Standardaufgaben. Durch Vermeidung von wiederholenden manuellen Tätigkeiten verringern sich Unaufmerksamkeiten und Flüchtigkeitsfehler. Auch die Kreativität des Softwaretesters wird durch gewonnene Zeiträume gefördert. Durch die gewonnene Zeit kann der Tester sich auch auf andere Bereiche konzentrieren, wie z. B. das Testen von bisher ausgelassenen Softwareteilen. Des Weiteren wird sich dadurch die Testfallqualität erhöhen.

Testautomatisierung kann auch Risiken bergen. Fehler oder falsche Annahmen in der Spezifikation oder Testfallerstellung lassen sich mit der massenhaften und auch meist anonymen Durchführung von Testfällen schwer entdecken und haben zumeist eine größere Auswirkung durch evtl. Potenzierung. Bei einem manuellen Test wird jeder Schritt im Testprozess beobachtet und bewertet. Wenn z. B. der gesamte Testprozess automatisiert ist, hat der Tester nur noch eingeschränkte Beobachtung und Kontrolle. Durch Testautomatisierung lassen sich Kennzahlen leichter erheben, weil mehr Möglichkeiten dafür vorhanden sind. Durch schlechte Algorithmen oder falsche Interpretationen können Kennzahlen aber ein falsches Bild über die Testergebnisse und somit über die Softwarequalität vermitteln.

Änderungen am Testobjekt führen bei automatisierten Tests oft zu hohem Anpassungsaufwand des Automatisierungskonzepts. Dies betrifft vor allem komplexe, heterogene Testobjekte. Dieser Aufwand kann schnell den sonst manuellen Testaufwand übersteigen. Auch die technischen Anforderungen an den Tester erhöhen sich. Neben den bisher erforderlichen fachlichen Kenntnissen benötigen Tester zunehmend IT-Kenntnisse. Bei Erstellung eines Automatisierungskonzepts ist immer die Teilnahme von fachlichem sowie technischem Personal notwendig.

2.3 Der Testprozess

Der allgemeine Testprozess gliedert sich in folgende Einzelschritte:

- Testplanung
- Testfallerstellung
- Generierung von Testdaten
- Zuordnung von Testdaten zu Testfällen
- Testdurchführung
- Testfallmonitoring
- Testreport
- Säuberung

Für einen vollständigen Testprozess sind alle oben genannten Einzelschritte notwendig, da bspw. ein Testreport nicht ohne Testfallmonitoring erstellbar ist. Bis auf die Testdurchführung mit parallelem Monitoring der durchgeführten Testfälle laufen alle Einzelschritte sequenziell ab. In der Testplanung erfolgt eine neue Beurteilung der Lage. Die Beurteilung entsteht aus der Interpretation der Testergebnisse des Testreports sowie der dazu aktuell vorliegenden Randbedingungen. Der Testfall ist i. Allg. die Kombination eines Ist-Zustands (Beschreibung mit Stammdaten) und eines Initialprozesses zur Zustandsänderung (Beschreibung mit Bewegungsdaten). Der Testfall muss zusätzlich eine Beschreibung des zu erwarteten Ergebnisses besitzen. Mit dieser Information kann im Testfallmonitoring eine Bewertung stattfinden, indem das erwartete Ergebnis mit dem tatsächlichen Ergebnis verglichen wird. Die so gewonnenen Informationen können anschließend aufbereitet und in einem Testreport z. B. graphisch dargestellt werden. Hier ist dann eine Aggregierung der

Informationen empfehlenswert. Das erleichtert die Auswertung der gesamten Testvorgänge und ist die Grundlage für eine evtl. Neuausrichtung in der Testplanung. Geeignete Testmetriken dazu werden im Kapitel 2.6.3 behandelt. Zum Schluss muss eine Säuberung des Testobjekts erfolgen. Die Säuberung soll das Testobjekt für den nächsten Testzyklus vorbereiten. Dies ist erforderlich, weil Testfälle eindeutige Vorbedingungen definieren. Werden diese Vorbedingungen nicht erfüllt, ist der Testfall entweder nicht durchführbar oder das Testergebnis wird verfälscht. Eine Möglichkeit der Säuberung besteht schon im Nachbereiten eines einzelnen Testfalls, indem alle Änderungen am Testobjekt rückgängig gemacht werden. Damit wird der Ursprungszustand wiederhergestellt. Außerdem kann auch ein vor dem Testprozess erstellter Snapshot des gesamten Testsystems neu eingespielt werden.

Die Abbildung 1 stellt den allgemeinen Testprozess als UML-Aktivitätsdiagramm dar. Die Notationsregeln wurden von WALDEMAR CZUCHRA [3] übernommen.

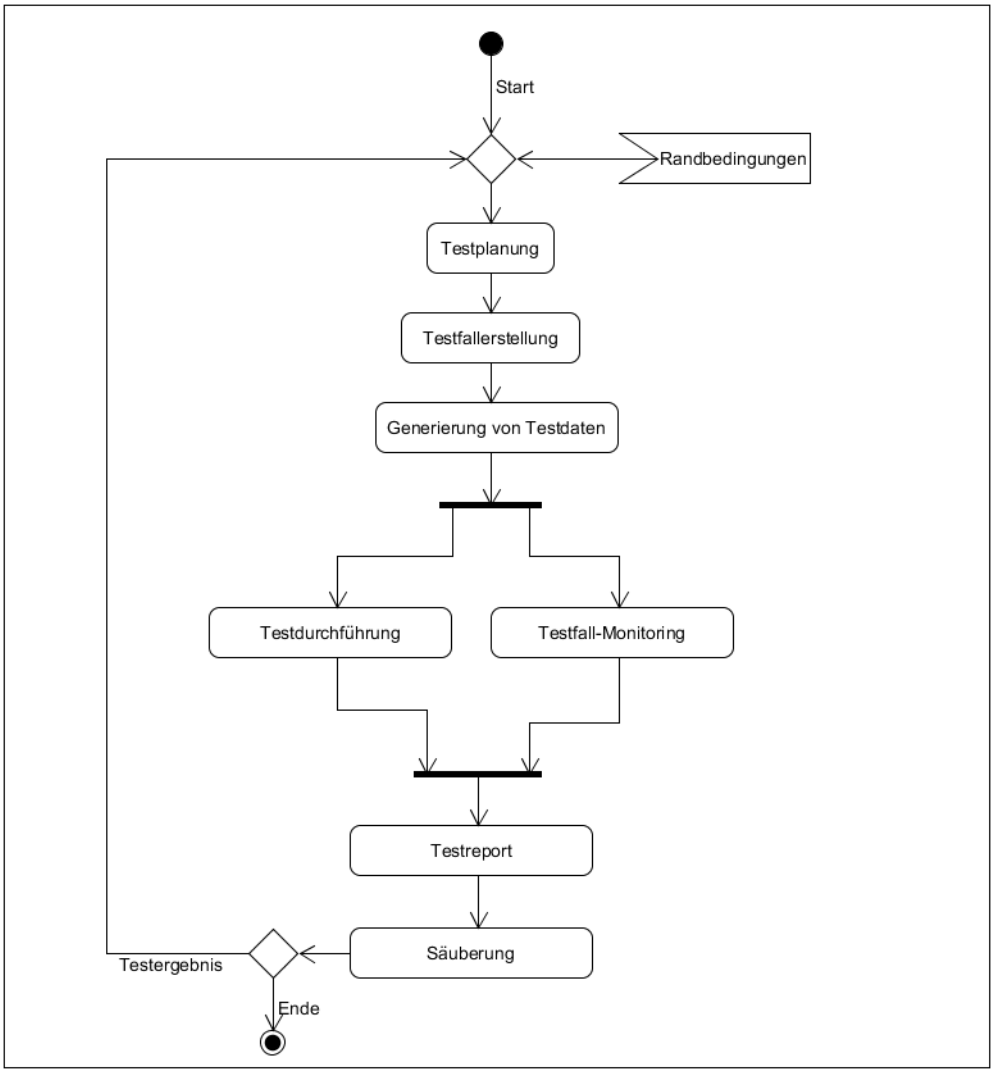

Abbildung 1: Allgemeiner Testprozess

Die oben genannten Teilprozesse lassen sich in weitere Teilprozesse (Teilaktivitäten) zerlegen. Eine Übersicht dazu liegt in der Anlage Teil 2 vor. Des Weiteren sind alle Teilprozesse theoretisch automatisierbar.

2.4 Technische Konzepte

2.4.1 Graphische Benutzeroberfläche (GUI)

Ein GUI-Testwerkzeug bezieht sich zur Überprüfung der Systemfunktionen auf die Benutzeroberfläche (GUI, engl. graphical user interface) des Testobjekts. Der GUI-Test ist damit auf der höchsten Testebene (Benutzerebene) angesiedelt. Durch den Oberflächentest werden indirekt die dahinterliegenden Systemfunktionen bzw. Datenbanken geprüft. Die Testergebnisse werden daher aus realistischen Interaktionen des Testobjekts mit dem Benutzer generiert. Eine Fehlerortung ist durch die hoch angesetzte Testebene allerdings sehr erschwert (siehe dazu Kapitel 2.5.3). Unter GUI-Testwerkzeuge fallen z. B. Testroboter, in der Literatur auch Capture & Replay Tools genannt. Diese Tools zeichnen die Interaktionen des Benutzers mit der Programmoberfläche auf. Die Aufzeichnungen können dann, mit Parametern ergänzt, beliebig oft wiederholt werden.

2.4.2 Anwendungsprogrammierschnittstelle (API)

Die API (engl. application programming interface) - auch Anwendungsprogrammierschnittstelle genannt - ist eine strukturierte und beschriebene Schnittstelle einer Anwendung, die anderen Anwendungen eine Programmierung / Parametrierung der bereitstellenden Instanz ermöglicht. Eine Anwendungsprogrammierschnittstelle ist strukturiert und damit maschinenlesbar. Des Weiteren muss eine Beschreibung zu deren Verwendung vorliegen. Unter Anwendungsprogrammierschnittstellen lassen sich z. B. Datenbanken, Webservices sowie EDI (engl. electronic data interchange) zusammenfassen. EDI umfasst viele unterschiedliche Nachrichtenstandards wie z. B. EDIFACT oder XML.

„In einer eher technischen Logik sind APIs also wie das maschinelle Äquivalent zum User-Interface, welches für Menschen optimiert wurde und so „menschenlesbar“ ist. Die API ist eine für Software zugeschnittene Schnittstelle, also maschinenlesbar. Das Application-Programming-Interface ermöglicht einen klar abstrahierten und strukturierten Zugriff auf die Funktionen des Backends. Darüber können Daten beispielsweise in einer besonders gut weiterzubearbeitenden und reduzierten Form ausgetauscht werden.“ [4]

Tatsächlich gibt es verschiedene Perspektiven bei der Definition von Anwendungsprogrammierschnittstellen und graphischer Benutzeroberflächen. Zum Beispiel ist eine für Webservices verwendete XML-Datei auch für den Menschen lesbar. Mit Testrobotern lassen sich auch Benutzeroberflächen maschinell bedienen und auswerten.

2.4.3 Mock-Objekte und Service-Virtualisierung

Besonders auf der Ebene des Modultests kommen Mock-Objekte (Deutsch: Attrappen) zum Einsatz. Mocks simulieren im Softwaretest das Verhalten von echten Objekten, indem diese definierte Werte an den Anwendungsprogrammierschnittstellen bereitstellen. Dieses Vorgehen bietet sich z. B. für das Simulieren von Schnittstellen des zu testenden Objekts an.

„Mocks können für den automatisierten Test mehrere Zwecke erfüllen. Zusätzlich zur häufigen Verwendung für das Vereinfachen von Testszenarien im Unit Test oder Integrationstest können Sie auch im Systemtest in vielen Fällen eingesetzt werden: von der Isolation von zu testenden Komponenten über die Simulation von Fremdsystemen und Systemen, für die kein Testsystem zur Verfügung steht, bis hin zum Test von Komponenten auf Robustheit durch die Rückgabe von ungültigen Werten oder andere fehlerhafte Verwendung von Schnittstellen durch äußere Komponenten und Systeme. Auf höheren Testebenen (also System- oder Systemintegrationstests) gibt es in vielen Fällen ähnliche Konstrukte: Testtreiber und Testrahmen für Integrations- und Systemtests verwenden ähnliche Prinzipien.“ [2] S. 69

Das gleiche Prinzip erfüllt auch die Service-Virtualisierung. Hier werden aber nicht nur Schnittstellen imitiert, sondern ganze Softwarekomponenten in einer heterogenen Softwarelandschaft. Im EDIFACT-Verkehr ist zudem die Simulation von Marktpartnern möglich. Der Vorteil von simulierten Marktpartnern ist deren bestimmbares und vorhersagbares Verhalten. Das gibt eine bessere Kontrolle über den Testprozess. Anderseits werden manche Fehler nur durch zufällige, vorher nicht definierte Aktionen hervorgerufen.

2.5 Logische Konzepte

2.5.1 Datengetriebene Testfalldarstellung

Das Prinzip der datengetriebenen Testfalldarstellung ist die Trennung der Testfälle von den Testdaten. Der Testfall selbst ist zunächst nur eine beschreibende Struktur (Klasse). Erst mit entsprechenden Daten wandelt sich der Testfall als abstrakte beschreibende Struktur zu einem durchführbaren Testfall (Objekt). Die Daten können z. B. in Tabellenkalkulationsprogrammen oder Datenbanken hinterlegt sein. Auch ein direktes Abgreifen der Daten aus dem Testobjekt ist möglich. Mit diesem Konzept kann leichter auf Änderungen des Testobjekts reagiert werden, indem nur noch die Testfallbeschreibung angepasst werden muss. Auch die Säuberung des Testobjekts wird vereinfacht. Bereits verwendete Daten können z. B. mit Hilfe von schwarzen Listen der weiteren Benutzung entzogen werden. In der Abbildung 2 ist das Prinzip der datengetriebenen Testfalldarstellung bildlich dargestellt.

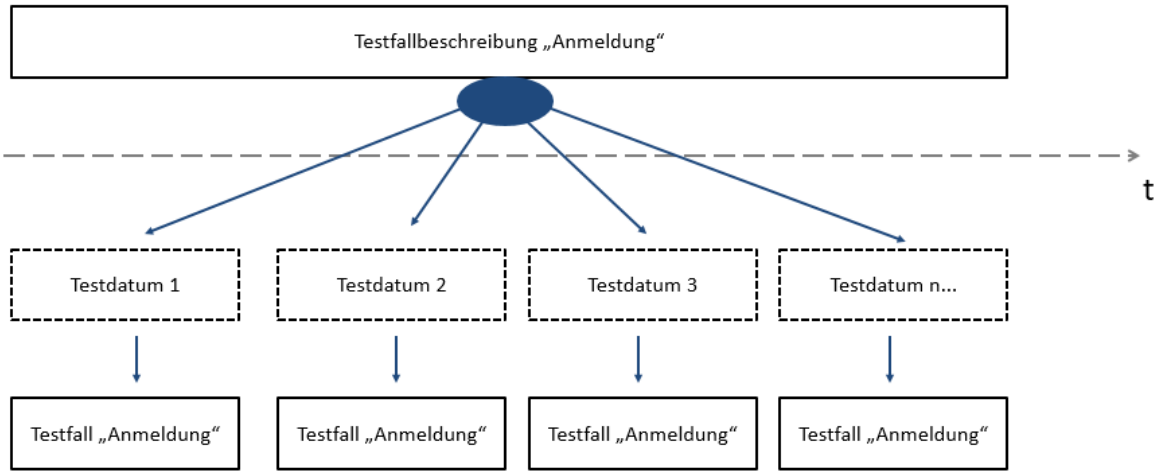

Abbildung 2: Datengetriebene Testfalldarstellung

2.5.2 Schlüsselwortgetriebene Testfalldarstellung

Ein weiteres Konzept ist die Kombination von einzelnen möglichst atomaren Testschritten zu komplexeren Testfällen. Das verringert den Anpassungsaufwand bei strukturellen Veränderungen des Testobjekts. Folgendes Beispiel soll das Prinzip näher erläutern.

Beschreibung des Testfalls:

Im Testobjekt robotron*ecount soll ein neuer Marktpartner angelegt werden. Zusätzlich wird diesem Marktpartner ein Ansprechpartner mit Name, Vorname, Telefonnummer und Emailadresse hinterlegt. Anschließend soll eine Löschung des angelegten Konstrukts erfolgen.

Zunächst werden aus der textlichen Beschreibung modulare Schritte erstellt:

1. Programm starten
2. Einloggen
3. Menüpunkt auswählen
4. Marktpartner anlegen
5. Ansprechpartner anlegen
6. Name, Vorname, Telefonnummer und Emailadresse zuweisen
7. Ansprechpartner löschen
8. Marktpartner löschen
9. Ausloggen
10. Programm schließen

Eine zusätzliche Möglichkeit ist die Einteilung der Module in Klassen bezüglich der Testfallvorbereitung, Testfalldurchführung und Testfallnachbereitung. Die Module 1, 2 und 3 sind

voraussetzende Aktionen. Diese dienen sozusagen nur zur Testfallvorbereitung. Die Module 4 bis 8 bilden den konkreten Testfall ab. Die Module 9 und 10 dienen zur Testfallnachbereitung, mit der ein definierter Endzustand erreicht wird. Die Testfallnachbereitung ist auch eine Vorsäuberung des Testobjekts. Trotzdem sollte nach jedem Testzyklus (Testrelease) eine globale Säuberung im Testobjekt stattfinden (Vgl. Kapitel 2.3).

Bereich	Modul	Beschreibung
	1.	Programm starten
	2.	Einloggen
	3.	Menüpunkt auswählen
	4.	Marktpartner anlegen
	5.	Ansprechpartner anlegen
	6.	Name, Vorname, Telefonnummer und Emailadresse zuweisen
	7.	Ansprechpartner löschen
	8.	Marktpartner löschen
	9.	Ausloggen
	10.	Programm schließen

Tabelle 1: Testmodule

Die gebildeten modularen Aktionen (Module) können mit gewissem Anpassungsaufwand nun auch anderen Testfällen zur Verfügung stehen. Zum Beispiel benötigt fast jeder Testfall die zwei Module „Programm starten“ und „Einloggen“. Hier bietet sich ein Katalog mit möglichst rudimentären Aktionen an. Testfälle können so leicht aus einer gegebenen Auswahl zusammengestellt werden. Auch der verfügbare Variantenreichtum erhöht sich dadurch. Im oben genannten Beispiel könnte man auch ein Negativ-Testfall erstellen, indem man die Schritte 7 und 8 miteinander vertauscht. Hier müsste dann in der Durchführung eine Fehlermeldung den Vorgang verhindern und darauf hinweisen, dass der Fremdschlüssel in der Ansprechpartner-Entität auf einen sonst nicht mehr vorhandenen Primärschlüssel der Marktpartner-Entität verweist (Referentielle Integrität).

Die Zerlegung der Testfälle in rudimentäre Teilaktivitäten vereinfacht die Testautomatisierung. Bei Änderungen am Testobjekt oder des Testkonzepts müssen dann nur noch die Module einzeln angepasst werden. Durch die Übergabe von Parametern an die Module lassen sich die beiden Konzepte der datengetriebenen und schlüsselwortgetriebenen Testfalldarstellung gut miteinander kombinieren.

2.5.3 Testebenen von Software

Software lässt sich auf verschiedenen Ebenen testen. In der Literatur gibt es keine einheitliche Bezeichnung für dieses Konzept. Neben dem Begriff Testebene gibt es auch die Begriffe Prüfebenen[1] und Teststufen[2]. Allen gemeinsam ist die Unterteilung der Programmstruktur einer Software in folgende Ebenen:

- Modulebene
- Integrationsebene
- Systemebene

Nach den Ebenen der Programmstruktur, wobei die Systemebene über der Integrationsebene und die Integrationsebene über der Modulebene liegen, lassen sich nun die Testfälle nach Klassen zuordnen. Im Folgenden soll diese Einteilung als Testebene bezeichnet werden:

- Unit-Test
- Integrationstest
- Systemtest
- Abnahmetest

[1] [5] S. 159

[2] [18]

Bei der Entwicklung einer neuen Software ergeben sich diese Testebenen automatisch. Dies ergibt sich aus einem strukturierten Entwicklungsprozess. Die Planung einer Software erfolgt aus der Top-down-Sicht. Zunächst werden alle funktionalen und nicht funktionalen Eigenschaften der zukünftigen Software beschrieben. Daraus ergeben sich Schritt für Schritt Beschreibungen für Funktionen und Teilfunktionen. Dieser Vorgang wird Dekomposition genannt. Die Umsetzung der Beschreibungen läuft dann aus der Bottom-up Sicht ab. Zuerst werden einfache Funktionen erstellt. Anschließend werden die einzelnen Funktionen miteinander verbunden und in Klassen vereint. Klassen werden in der Literatur auch als Units, Komponenten oder Module bezeichnet. Die Klassen werden dann zu Paketen bzw. Klassenbibliotheken zusammengefasst. Dieser Vorgang wird Aggregation genannt.

„Mit dem Begriff Unit wird eine atomare Programmeinheit bezeichnet, die groß genug ist, um als solche eigenständig getestet zu werden. Diese recht vage formulierte Begriffsdefinition lässt einen erheblichen Spielraum zu und in der Praxis unterscheiden sich zwei atomare Programmeinheiten oft beträchtlich in ihrer Größe. Hier können sich Units von einzelnen Funktionen, über Klassen, bis hin zu Modulverbünden in Form von Paketen und Bibliotheken erstrecken. Aus diesem Grund wird der Unit-Test nicht selten auch als Modultest oder noch allgemeiner als Komponententest bezeichnet. Wird die Methodik des Unit-Tests auf größere Klassen- oder Modulverbünde angewendet, so geht der Testprozess fast nahtlos in den Integrationstest über […]. Die klare Trennlinie, die in der Literatur zwischen den beiden Testarten gezogen wird, ist in der Praxis an vielen Stellen durchbrochen.“ [5] S. 159

Im Gegensatz zum Unit-Test und Integrationstest liegt der Systemtest und Abnahmetest auf gleicher Ebene.

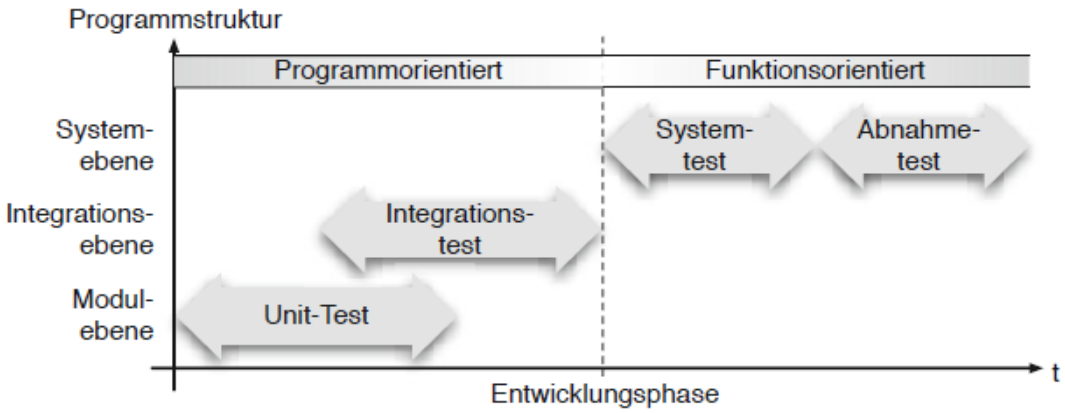

Abbildung 3: Die vier Prüfebenen des Softwaretests [5] S. 159

Doch warum sollen auch bei bestehender Software verschiedene Teststufen Anwendung finden? Bei hohen Testebenen können grundsätzlich mehr Fehler entdeckt werden. Die Fehlerursache aber ist zumeist schwerer zu finden. Der Grund liegt an der Komplexität des

Testobjekts. Bei hoher Testebene, wie z. B. beim Funktionstest werden bereits interagierende Module getestet. Dies hat zur Folge, dass ein Fehler nicht eindeutig zugeordnet werden kann (Fehler-Ortbarkeit). Im Modultest werden alle Komponenten einzeln geprüft, wobei auch die Fehler besser zugeordnet werden können. Hier findet dann aber keine Prüfung der Interaktionen aller Gesamtmodule statt. In niedrigen Testebenen sind demnach nicht alle Fehler des Gesamtsystems enthalten. Diese Feststellung soll im Folgenden mit dem Begriff „Fehlerverfügbarkeit“ bezeichnet werden.

Dazu folgendes theoretisches Beispiel: In der Abbildung 4 wird eine abstrakte Darstellung der Programmebenen verwendet. Des Weiteren werden für dieses Beispiel vier aufeinander aufbauende Testebenen angenommen. Die Fehlerstellen sind mit grünen Punkten markiert.

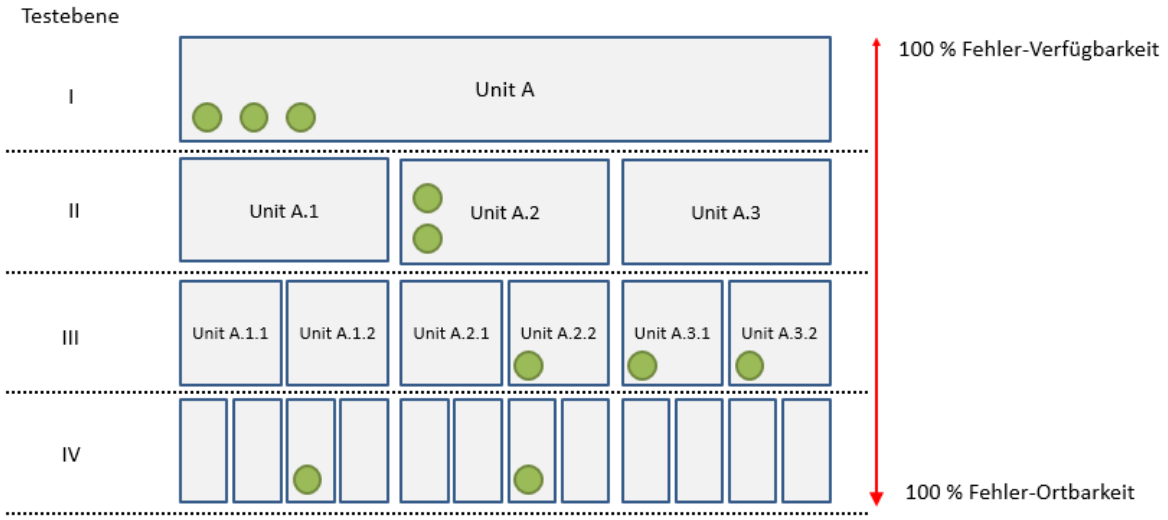

Abbildung 4: Programmebene und Fehlerrelation

Die Gesamtzahl der Fehler im System beträgt zehn. In der Testebene römisch 3 können aber nur fünf Fehler ermittelt werden. Das ergibt eine Fehlerverfügbarkeit von 50%. Mit einer Fehlerfindungsquote von 100% können daher nur die Hälfte aller Fehler im System gefunden werden. In der Praxis liegt zudem die Fehlerfindungsquote bedeutend niedriger. Die Fehler-Ortbarkeit ergibt sich aus der Anzahl von kleinstmöglichen (atomaren) Bestandteilen einer untersuchten Einheit. Im Fall von Softwarecode ist die kleinste Einheit eine Anweisung. Nach P. FISCHER und P. HOFER ist eine Anweisung eine einzelne, in einer Programmiersprache befohlene und den Zustand von Daten oder Parametern verändernde Aktivität. [6] S. 41

Vor Auslieferung von neuen oder geänderten Funktionen erfolgen schon vom Auslieferer robotron sogenannte Unit-Tests der betroffenen Codestellen. Im Fachbereich erfolgt zusätzlich ein Funktionstest. Zum großen Teil werden EDIFACT-Dateien als Träger bzw. Trigger der Testfall-Parameter benutzt. Dies ermöglicht eine weitestgehend realistische Nachbildung des realen Marktverkehrs. Eine grundlegende Beschreibung des EDIFACT-Formates ist im Kapitel 3 der Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ [1] zu finden. Dort ist das Subset UTILMD näher erklärt.

2.6 Werkzeuge zur Testautomatisierung

Testwerkzeuge lassen sich in die Komponenten Testplanung, Testdurchführung und die Art der Fehlererkennung einteilen. Die Testplanung beinhaltet das Testfalldesign sowie das Ergebnismanagement. Die Themengebiete „Fehlermanagement“ und „Reports (Statistik)“ werden nicht näher behandelt. Viele Testwerkzeuge vereinigen mehrere Komponenten.

2.6.1 Fehlererkennung

Die Fehlererkennung erfolgt durch eine dynamische bzw. statische Programmanalyse. Der Begriff „Programmanalyse“ ist gleichzusetzen mit dem dynamischen und statischen Testverfahren.

„Voraussetzung dynamischer Software-Testverfahren ist die Lauffähigkeit des Prüflings. Der Test findet im laufenden Prozess (Programmlebensphase) des Prüflings statt. Dazu muss eine entsprechende Performance des Prüflings vorliegen, d. h. die Antwortzeiten des Prüflings müssen annehmbar sein. Es dürfen auch keine Laufzeitfehler auftreten. Fehler werden durch den Vergleich von Ein- und Ausgabedaten ermittelt. Dynamische Testverfahren wären z. B. der White-Box-Test, der Black-Box-Test und der Regressionstest. […]

Das statische Testverfahren erfolgt i. d. R. bereits beim Softwarehersteller. Der Prüfling wird beim Test nicht ausgeführt. Dies betrifft die Entwicklungs- und Kompilierungszeit. Ein Beispiel dafür ist die Syntaxprüfung der vorliegenden Code-Zeilen oder die Prüfung während der Übersetzungszeit (Kompilierungszeit) des Prüflings durch den Compiler. Des Weiteren können auch formale Software-Testverfahren dem statischen Software-Testverfahren zugeordnet werden.“ [1] S. 9

Eine nähere Beschreibung und fachliche Vertiefung von dynamischen und statischen Testverfahren ist in der Projektarbeit „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ [1] im Kapitel 4.3 zu finden.

Testwerkzeuge lassen sich somit nach dynamischen oder statischen Testverfahren einordnen, wobei ein Testwerkzeug zumeist nur ein Testverfahren abdeckt. Testwerkzeuge für

statische Testverfahren prüfen das Softwaremodell wie z. B. das Datenmodell oder Anwendungsmodell. In diese Kategorie fallen z. B. Modellierungswerkzeuge mit eingebauten Konsistenzprüfungen. Beispiele dafür sind Microsoft Access, MySQL Workbench, Oracle Designer und diverse UML-Editoren. Tools zum Vergleich von Programmcode gehören ebenfalls in die Kategorie der statischen Testverfahren. Werkzeuge für dynamische Testverfahren sind z. B. Capture & Replay-Tools oder EDIFACT-Editoren. Die damit erstellten EDIFACT-Nachrichten lassen sich dann zur Laufzeit des Testobjekts (Prüfling) einsetzen.

2.6.2 Testfall-Design

Der Begriff Testfalldesign beschreibt die Art der Erzeugung von Testfällen. Die Grundlage des Testfalls bildet eine oder mehrere Anforderungen an das Testobjekt. Die Anforderung enthält konkrete Beschreibungen, die das Testobjekt erfüllen muss. Ein Testfall besteht aus Testdaten und der Testfallbeschreibung, welche sich wiederum aus Testaktivitäten und ein oder mehreren vergleichbaren Sollergebnissen zusammensetzt.

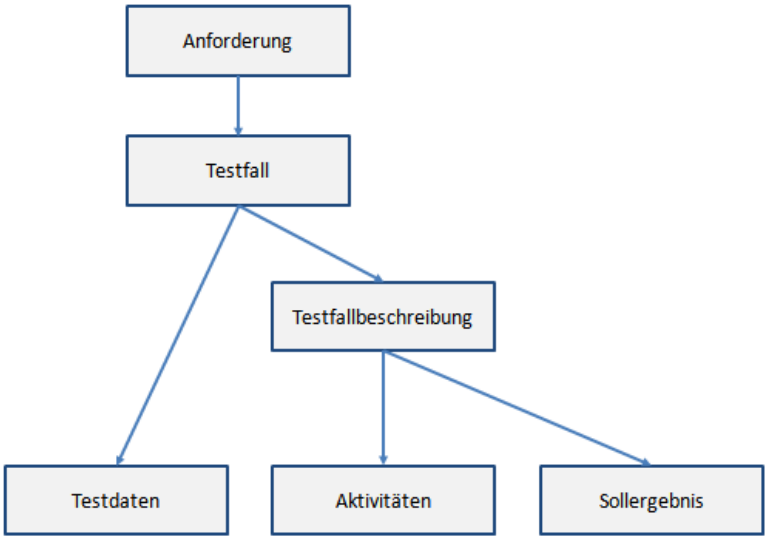

Abbildung 5: Testfalldesign

Eine weit verbreitete Methode des Testfalldesigns ist die Beschreibung des Testfalls in Textform. Grundsätzlich ist neben der Testdurchführung auch das Testfall-Design automatisierbar. Voraussetzung ist aber eine möglichst formal konkrete Beschreibung des Testfalls als Modell. Nach T. BUCSICS [2] S. 37-38 ergeben sich folgende Vorteile aus dem modellbasierten Testfalldesign:

- **Modular und abstrahierbar**
  - Während des Modelldesigns können durch die Bildung von Diagrammhierarchien und der Kapselung von Teilmodellen verschiedene Abstraktionsebenen geschaffen werden, wobei sich die verschiedenen Stakeholder auf die für sie interessanten Ebenen konzentrieren können.
- **Automatisierbar**
  - Durch die formale Darstellung im Modell können Werkzeuge Testfälle besser generieren, als dies bei textuellen Anforderungen der Fall ist.
- **Systematisch**
  - Die Testfallerstellung wird stark systematisiert. Dies geschieht einerseits durch festlegbare Vorgehensweisen bei der Modellierung, anderseits durch die Ableitung von Testfällen aus Modellen nach vorgegebenen Regeln.
- **Flexibel und sukzessive integrierbar**
  - Die Methodik kann sukzessive eingeführt werden. Beispielsweise wird das Modell zunächst für die Priorisierung und die Kommunikation zwischen den Beteiligten herangezogen. Später kann es um die Möglichkeiten der automatisierten Testfallgenerierung erweitert werden.
- **Verständlich und kommunizierbar**
  - Das Modell ermöglicht einen Überblick über das zu testende System und bietet eine verständliche Kommunikationsgrundlage, auch für Personen, die technisch nicht versiert sind. Modelle haben den Vorteil, dass sie gut lesbar sind.
- **Wartbar und wiederverwendbar**
  - Aufgrund der Wiederverwendbarkeit und der Toolintegration müssen bei einer Änderung nur wenige Modellelemente angepasst werden. Redundanzen werden dadurch vermieden und Impact-Analysen vereinfacht.

2.6.3 Testmetriken (Ergebnismanagement)

Grundsätzlich dienen Testmetriken (Kennzahlen) dem Ergebnismanagement. Testmetriken geben einen Überblick über konkrete Verhältnisse und Zustände im aktuellen Testprozess und sind die Grundlage für Teststatistiken (Reports). Automatisierte Testprozesse vereinfachen die Ermittlung von Kennzahlen.

Nach F. WITTE [7] S. 210 werden Metriken zur Verfolgung der Testprozesse unterteilt in:

- Testfallorientierte Metriken, z. B. Anzahl der Testläufe, Anzahl der erfolgreichen Tests, Anzahl der Testwiederholungen
- Fehlerbasierte Metriken, z. B. Anzahl der Fehlermeldungen
- Testobjektbasierte Metriken, z. B. Codeüberdeckung

Testmetriken helfen auch bei der Bestimmung vom Automationsgrad des Testprozesses. Mögliche Metriken sind z. B.:

- Anzahl der durchgeführten Testfälle pro Personenstunde
- Gesamtzahl der Testfälle geteilt durch die Anzahl der manuell erstellten Testfälle

Die gesammelten Informationen müssen anschließend aufbereitet und interpretiert werden, wobei bei der Interpretation oft Fallstricke lauern, weil z. B. die Komplexität der Testfälle nicht beachtet wird.

„[…] Eine Metrik „Anzahl der Testfälle“ z. B. vermengt umfangreiche, komplexe und langwierige mit einfachen und schnell durchzuführenden Testfällen. Wenn nun am Anfang des Projekts die einfachen, schnell durchzuführenden Testfälle durchgeführt werden, steigt die Kurve der Anzahl Testfälle schnell nach oben und verleitet dazu, sich zurücklehnen zu können. Aber wenn es zu den schwierigeren Testfällen kommt, lässt sich dieses Tempo nicht mehr durchhalten, und die Kurve nach oben wird flacher, obwohl sich an der Effizienz des Testprozesses und am Einsatz des Testteams gar nichts geändert hat. […]“ [7] S. 210

Doch wie kann die Komplexität eines Testfalls bestimmt werden? Im Kapitel 2.6.2 wurde der allgemeine Aufbau eines Testfalls erläutert. Ein Testfall besteht demnach aus beschreibenden Aktivitäten, Testdaten und einem Sollergebnis. Nach H. SNEED und S. JUNGMAYR drückt sich die Komplexität der Testfälle in vier Kennzahlen aus [8] S. 30:

- Testdatendichte
- Testdatenkomplexität
- Testfallvolumen
- Testfallintensität

Die vier Metriken sollen am Beispiel eines Anwendungsfalls interpretiert werden. Anschließend wird mit den Ergebnissen der Komplexitätsfaktor bestimmt. Für den Anwendungsfall liegt folgende Spezifikation vor:

Pseudocode der Spezifikation:

```
// Aufruf der Funktion Fnc_Spannungsebene
string Spannungsebene = Fnc_Spannungsebene(Spannung);

// Funktion Fnc_Spannungsebene
static string Fnc_Spannungsebene(Int Spannung)
{
    if (Spannung > 0 and Spannung <= 1000)
        {
        return „NS“
        }
    else if (Spannung > 1000 and Spannung <= 35.000)
        {
        return „MS“
        }
    else if (Spannung > 35.000 and Spannung <= 150.000)
        {
        return „HS“
        }
    else
        {
        return „Ungueltig“
        }
}
```

Der Anwendungsfall besteht aus einer Funktion. Es soll nun ein Schnittstellentest der Funktion stattfinden. Folgende Testfälle sind dafür notwendig:

Testfall-Nr.	Inputdaten (Integer)	Sollergebnis (String)
1	1.000	„NS“
2	35.000	„MS“
3	150.000	„HS“
4	0	„Ungueltig“

Tabelle 2: Testfälle für die Funktion Fnc_Spannungsebene

Damit liegen folgende Parameter vor:

- Input-Werte mit Datentyp Integer: 1000, 35000, 150000, 0
- Output-Werte mit Datentyp String: „NS“, „MS“, „HS“, “Ungueltig”

2.6.3.1 Testdatendichte

Ein Testfall besteht zumeist aus einer Sammlung von verschiedenen Datentypen wie z. B. Integer und String. Die Testdatendichte ist der Quotient aus der Anzahl an unterschiedlichen Testdatentypen zu allen vorliegenden Testdatenwerten am Testfall. Die Testdatenwerte werden im Folgenden als Testdaten bezeichnet.

$$Testdatendichte = \frac{Testdatentypen}{Testdaten}$$

Formel 1: Testdatendichte nach F. WITTE [7] S. 211

Unser Beispiel beinhaltet insgesamt zwei Testdaten pro Testfall. Diese teilen sich in zwei verschiedene Datentypen (Integer und String) auf.

$$Testdatendichte = \frac{2}{2} = 1$$

2.6.3.2 Testdatenkomplexität

Die Testdatenkomplexität ist der Quotient aus steuernden Testdaten zu allen vorliegenden Testdaten im Testfall. „[…] Steuernde Testdaten bestimmen den Verlauf des Testfalls. Nur wenn die Variablen definierte Werte annehmen, schlägt das System den beabsichtigten Pfad ein […]. Je mehr Steuerdaten vorliegen, desto komplexer wir der Test.“ [7] S. 213

$$Testdatenkomplexität = \frac{Steuernde\ Testdaten\ (Prädikate)}{Alle\ Testdaten}$$

Formel 2: Testdatenkomplexität nach F. WITTE [7] S. 212

Im genannten Beispiel liegen pro Testfall zwei Testdaten vor. Ein Testdatum bestimmt den Verlauf des Testfalls.

$$Testdatenkomplexität = \frac{1}{2} = 0{,}5$$

2.6.3.3 Testfallvolumen

Das Testfallvolumen beschreibt das durchschnittliche Verhältnis der Anzahl von eingesetzten Testfällen zu der Anzahl von sämtlichen Parametern, die auf den Test einwirken. Je mehr Parameter mit einem Testfall abgedeckt werden können, desto weniger Testfälle müssen erstellt werden. Dafür sind diese Testfälle aber komplexer und damit schwieriger zu erstellen und zu verwalten. Daher sollten die Testparameter mit möglichst vielen einfachen Testfällen angesprochen werden.

$$Testfallvolumen = 1 - \left(\frac{Testfälle}{Testparameter}\right)$$

Formel 3: Testfallvolumen nach H. SNEED und S. JUNGMAYR [8] S. 30

Im Beispiel beschreiben vier Testfälle das Testszenario. Die Testparameter sind die Summe der Argumente der Testfälle sowie die Rückgabewerte der Zielfunktion.

$$Testfallvolumen = 1 - \left(\frac{4}{8}\right) = 0{,}5$$

2.6.3.4 Testfallintensität

Die Testfallintensität beschreibt das Verhältnis von Testfallanzahl zu der Anzahl von zu testenden Funktionen (Zielfunktionen). Die Kennzahl beschreibt demnach die durchschnittliche Anzahl von Zielfunktionen, die ein Testfall durchläuft. Indirekt gibt dieser Quotient Aufschluss über die Testfallkomplexität unter der Voraussetzung, dass komplexere Testfälle mehr Zielfunktionen durchlaufen.

$$Testfallintensität = 1 - \left(\frac{Testfälle}{Zielfunktionen}\right)$$

Formel 4: Testfallintensität nach F. WITTE [7] S. 214

Im Beispiel liegt nur eine Zielfunktion vor. Dies würde den Gleichungswert über den Grenzwert 0 bringen und damit in den negativen Zahlenbereich übergehen. Daher wird stattdessen die Testfalldichte ermittelt.

2.6.3.5 Testfalldichte

Die Testfalldichte beschreibt das Verhältnis der Anzahl der Anwendungsfälle zur Anzahl der Testfälle. Im oben genannten Beispiel liegt ein Anwendungsfall vor (Spannungsebene ermitteln). Für einen vollständigen Test sind vier Testfälle nötig. Die Testfalldichte ist der reziproke Wert der Testfallintensität, wenn der Anwendungsfall mit genau einer Funktion abgebildet werden kann.

$$Testfalldichte = 1 - \left(\frac{Anwendungsfälle}{Testfälle}\right)$$

Formel 5: Testfalldichte nach H. SNEED und S. JUNGMAYR [8] S. 30

Im Beispiel wird eine Funktion mit vier Testfällen geprüft:

$$Testfalldichte = 1 - \left(\frac{1}{4}\right) = 0{,}75$$

2.6.3.6 Komplexitätsfaktor

Aus den vier ermittelten Komplexitätsmaßen lässt sich nun der Komplexitätsfaktor bestimmen. „Die aggregierte Testfallkomplexität ist der arithmetische Mittelwert der einzelnen Komplexitätsmaße. Der Komplexitätsfaktor zur Justierung der Testquantität ist die aggregierte Testfallkomplexität dividiert durch die mittlere Testfallkomplexität = 0,5." [7] S. 214

$$Komplexitätsfaktor = \frac{Aggregierte\ Testfallkomplexität}{Mittlere\ Testfallkomplexität}$$

Formel 6: Komplexitätsfaktor nach F. WITTE [7] S. 214

Unser Beispiel ergibt folgende Komplexitätsmaße:

Testdatendichte:	1
Testdatenkomplexität:	0,5
Testfallvolumen:	0,5
Testfalldichte:	0,75

$$Komplexitätsfaktor = \left(\frac{1 + 0{,}5 + 0{,}5 + 075}{4}\right) / 0{,}5 = 1{,}375$$

Der nun ermittelte Faktor erlaubt eine Gewichtung der Testfälle bezüglich deren Komplexität. Der Vorteil dieser Vorgehensweise liegt in der besseren Aufwandsabschätzung der durchzuführenden Testfälle, da nun eine Unterscheidung zwischen einfachen, schnell durchführbaren Testfällen und langwierigen, komplexen Testfällen möglich ist.

2.6.4 Testdurchführung

Die Testdurchführung beinhaltet jegliche Aktivitäten, die der Ausführung von bereitgestellten Testfällen dient. Oft sind im Testobjekt Zustandsänderungen nötig. Diese werden entweder über Eingaben in der Benutzeroberfläche, durch Programmierschnittstellen oder durch zeitliche Trigger (Fristen) eingeleitet. Dabei kann zwischen automatischen und manuellen Prozessschritten im Testobjekt unterschieden werden. Automatische Prozessschritte werden implizit vom Testobjekt ausgeführt. Um auch die manuellen Prozessschritte zu automatisieren, muss ein geeignetes Testwerkzeug ausgewählt werden. Bei Testrobotern wird die Aufgabe der Testdurchführung vom Testwerkzeug übernommen. Anders ist das bei Werkzeugen, die nur ausführbare Testfälle bereitstellen. Hier muss eine zusätzliche Instanz die manuellen Prozessschritte bedienen.

Nach T. BUCSICS und M. BAUMGARTNER müssen eine Rückverfolgbarkeit des konkreten Testfalls und deren Ergebnisse zu den logischen Testfällen und den Anforderungen gewährleistet sein [2] S. 17. Daher muss parallel zur Testdurchführung auch eine Protokollierung der instanziierten Testfälle und dessen verursachten Ergebnisse erfolgen. Dies kann z. B. durch Instrumentierung der Schnittstellen realisiert werden.

3 Analyse der System- und Testlandschaft

Im folgenden Kapitel wird das Testobjekt robotron*ecount analysiert. Als Analysewerkzeuge werden einige ausgewählte Architektur-Sichten des 4+1 Sichtenmodells verwendet. Die Analyse erfolgt in der Top-down-Sicht. Daher wird zunächst die Umgebung des Testobjektes betrachtet. Anschließend wird das Testobjekt selbst behandelt. Das erarbeitete Wissen dient als Grundlage des Testkonzeptes.

3.1 4+1 Sichtenmodell

Im Kapitel 2.5.3 wurde die zu testende Software aus der Perspektive der Testebene betrachtet. Eine weitere Perspektive sind Architektursichten bzw. Architektur-Frameworks wie z. B. das 4+1 Sichtenmodell, RM-ODP (Reference Model for Open Distributed Processing)[3] oder das Zachman-Framework[4], wobei das 4+1 Sichtenmodell aus dem Meta-Sichtenmodell RM-ODP spezifiziert wurde. Das Ziel eines Sichtenmodells ist die genaue Analyse des Testobjekts. Aus den Erkenntnissen können anschließend passende Testfälle und Testmethoden ermittelt werden. Im Folgenden soll die Erläuterung und Beschreibung des 4+1 Sichtenmodells erfolgen. Die gewonnenen Erkenntnisse werden anschließend zur Analyse des Testobjekts robotron*ecount angewendet.

Das 4+1 Sichtenmodell wurde von P. KRUCHTEN [9] entwickelt und sah zunächst fünf Architektur-Sichten vor. Später kam durch die Interpretation von C. LARMAN die Datensicht (data view) hinzu [10] S. 501-502. Die Bezeichnung 4+1 Sichtenmodell ist jedoch erhalten geblieben und umfasst folgende Komponenten [11] S. 99:

- **Anwendungsfallsicht (englisch: use-case view)**: Diese Architektur-Sicht ist zentral im 4+1-Sichtenmodell. Das 4+1-Sichtenmodell besagt, dass sämtliche architektonischen Entscheidungen auf den Anwendungsfällen des betroffenen Systems beruhen müssen. Diese Architektur-Sicht umfasst die wichtigsten Anwendungsfälle und dient als Basis und zur Validierung der anderen Architektur-Sichten.

- **Logische Sicht (englisch: logical view)**: In dieser Architektur-Sicht wird die Umsetzung der funktionalen Anforderungen betrachtet. Hier werden die wichtigsten Systembausteine (Subsysteme, Komponenten, Klassen etc.) und ihre Interaktionen behandelt.

[3] ISO-Norm ISO/IEC 10746; Metamodell zur Beschreibung von Informationssystemen. [17]

[4] Domänenneutraler Ordnungsrahmen zur Entwicklung von Informationssystemen. [16]

- **Implementierungssicht (englisch: implementation view)**: Diese Architektur-Sicht behandelt die Organisation und Verwaltung der statischen Artefakte (Quelltext, Grafiken etc.) in Pakete, Schichten etc.

- **Datensicht (englisch: data view)**: In dieser Architektur-Sicht werden Datenmodelle beschrieben und die Abbildung (englisch: mapping) zwischen Systembausteinen und persistenten Daten betrachtet.

- **Prozesssicht (englisch: process view)**: Verhalten und Verteilung des Systems zur Laufzeit sind die Themen dieser Architektur-Sicht. Parallelverarbeitung und konkurrierende Zugriffe stehen dabei im Mittelpunkt.

- **Verteilungssicht (englisch: deployment view)**: Diese Architektur-Sicht beschreibt, wie die statischen Artefakte aus der Implementationssicht physikalisch verteilt werden.

In den folgenden Gliederungspunkten soll das Testobjekt robotron*ecount sowie die Umgebung mit einer Auswahl eingangs genannter Sichten näher erläutert werden. Die Notationsregeln bei der Verwendung von UML-Diagrammen wurde von W. CZUCHRA [3] übernommen.

3.2 Systemlandschaft

Das Testobjekt wurde zum Teil schon in der Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ [1] untersucht. Im Folgenden soll eine Vertiefung der dort ermittelten Ergebnisse vorgenommen werden. Um die Funktionsweise vom Testobjekt robotron*ecount zu verstehen, erfolgt zunächst eine Analyse der gesamten Systemlandschaft. Anschließend wird das Testobjekt robotron*ecount näher betrachtet. Die Abbildung 6 stellt die Beziehungen zwischen den einzelnen Teilsystemen und Schnittstellen der Systemlandschaft dar. Als Grundlage wurde ein bereits vorliegendes Blockbild der Systemlandschaft aus der Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ [1] S. 33 verwendet.

"Nicht für alle Schnittstellen liegt die Verantwortung bezüglich des Managements innerhalb der Abteilung Netzzugangsmanagement. Daher wurden nur die Schnittstellen/Systeme ausgewählt, die in unmittelbarer Nähe des Teilsystems robotron*ecount liegen.“ [1] S. 17

Des Weiteren wird das Teilsystem NETAN nicht näher betrachtet. Im NETAN werden die Netzanschlüsse, Zählpunktkonstrukte und Netzverträge von Leistungskunden verwaltet. Die dort vorliegenden Anwendungsfälle korrelieren nicht direkt mit den Geschäftsprozessen der GPKE, GeLi Gas, WiM und MPES.

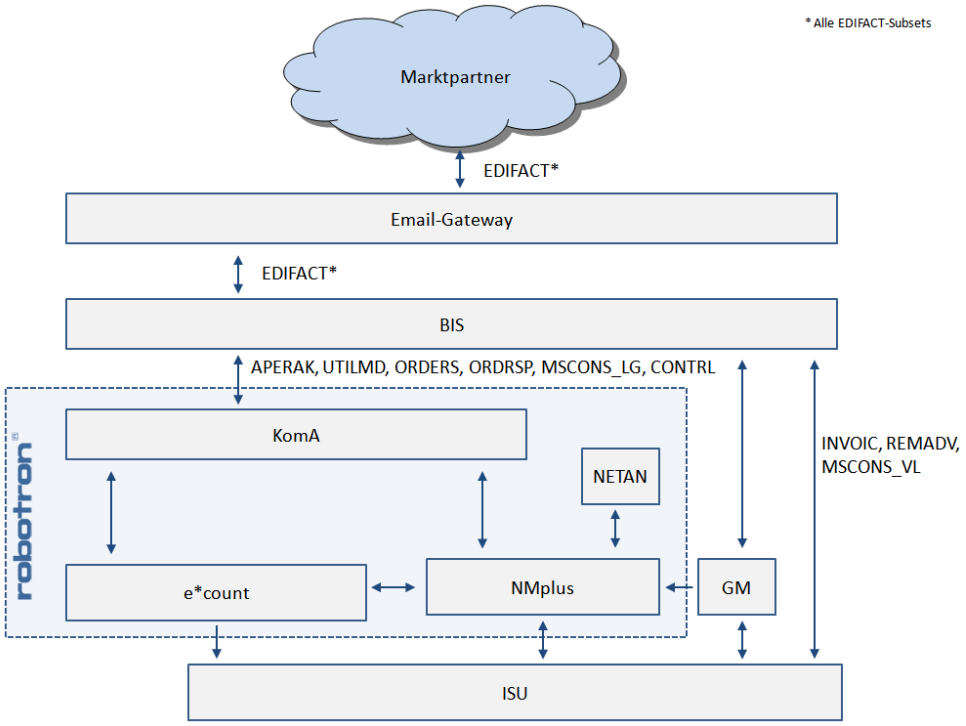

Abbildung 6: Logische Sicht der Systemlandschaft

Sofern keine EDIFACT-Subsets angegeben sind, erfolgt die Datenübertragung über andere Anwendungsschnittstellen wie z. B. Tabellen. In der Tabelle 3 sind die Aufgaben der genannten Teilsysteme näher beschrieben.

System	Name	Hersteller / Dienstleister	Aufgabe
Email-Gateway	CompuMail Gateway	Compumatica / GISA	• Verschlüsselung und Signierung von Emails zur sicheren Übertragung von EDIFACT-Nachrichten
BIS	Business Integration Server	Seeburger / GISA	• Verteilung der EDFACT-Nachrichten an die Umsysteme • Überprüfung auf CONTRL-Eingang
KomA	Kommunikationsautomatisierung	Robotron / GISA	• Generierung von EDIFACT-Dateien aus Systemtabellen sowie Füllung von Tabellen aus empfangenen EDIFACT-Dateien • Aufteilung der Nachrichten in Meldungen • Syntaxprüfung
GM	Gerätemanagement	innogy SE / GISA	• Verwaltung von technischen Gerätedaten wie z. B. Zähler, Schaltuhren, Kommunikationseinrichtungen sowie Strom- und Spannungswandler
NETAN	Netzanschlüsse	Robotron / GISA	• Verwaltung von Netzanschlüssen, komplexen Zählpunktkonstrukten sowie Netzverträgen

e*count	Energiedatenmanagement	Robotron / GISA	• Energiemengenbilanzierung und Verwaltung von Lastgangdaten • Marktpartnerverwaltung
NMplus	Netzzugangsmanagement	Robotron / GISA	• Abwicklung der GPKE, GeLi Gas, WiM und MPES
IS-U	Industry Solution Utilities	SAP / GISA	• Abrechnung der Netznutzung • Verwaltung von Zählerständen

Tabelle 3: Teilsysteme

Das Zusammenspiel der Systemkomponenten soll am Beispiel einer konkreten Ausprägung des GPKE-Prozesses Lieferbeginn mit anschließender Bereitstellung von Messwerten sowie Rechnungslegung der Netznutzung nach [12] näher erläutert werden. Die Teilprozesse Lieferbeginn, Messwertübermittlung und Netznutzungsabrechnung besitzen weitere Verzweigungen wie z. B. den Prozessschritt 3b (Abmeldungsanfrage) im Prozess Lieferbeginn. Diese zusätzlichen Verzweigungen sollen aufgrund der zusätzlichen Komplexität des Beispiels nicht näher erläutert werden. Die Abbildung 7 stellt die genannte Prozessausprägung als Sequenzdiagramm dar.

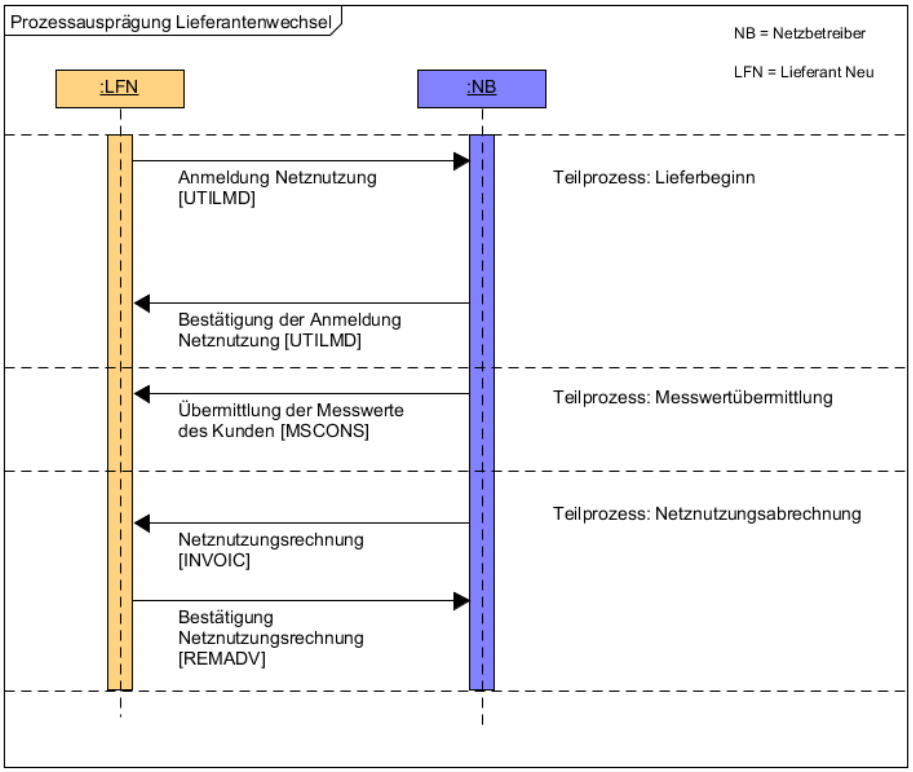

Abbildung 7: Prozessausprägung Lieferantenwechsel

Die Initialisierung der Prozessausprägung erfolgt durch eine Anmeldung der Netznutzung durch den neuen Lieferanten (LFN). Nach positiver Überprüfung durch den Netzbetreiber (NB) erfolgt eine Bestätigung der Anmeldung. Daran schließt auf Seite des Netzbetreibers die Messwertübermittlung an den neuen Lieferanten an. Die Messwerte sind die Grundlage der darauffolgenden Rechnungslegung der Netznutzung. Schlussendlich bestätigt der neue Lieferant die Netznutzungsrechnung.

In der Abbildung 8 wird der Prozess aus dem Sequenzdiagramm gemäß Abbildung 7 nun mit den beteiligten Systemen dargestellt.

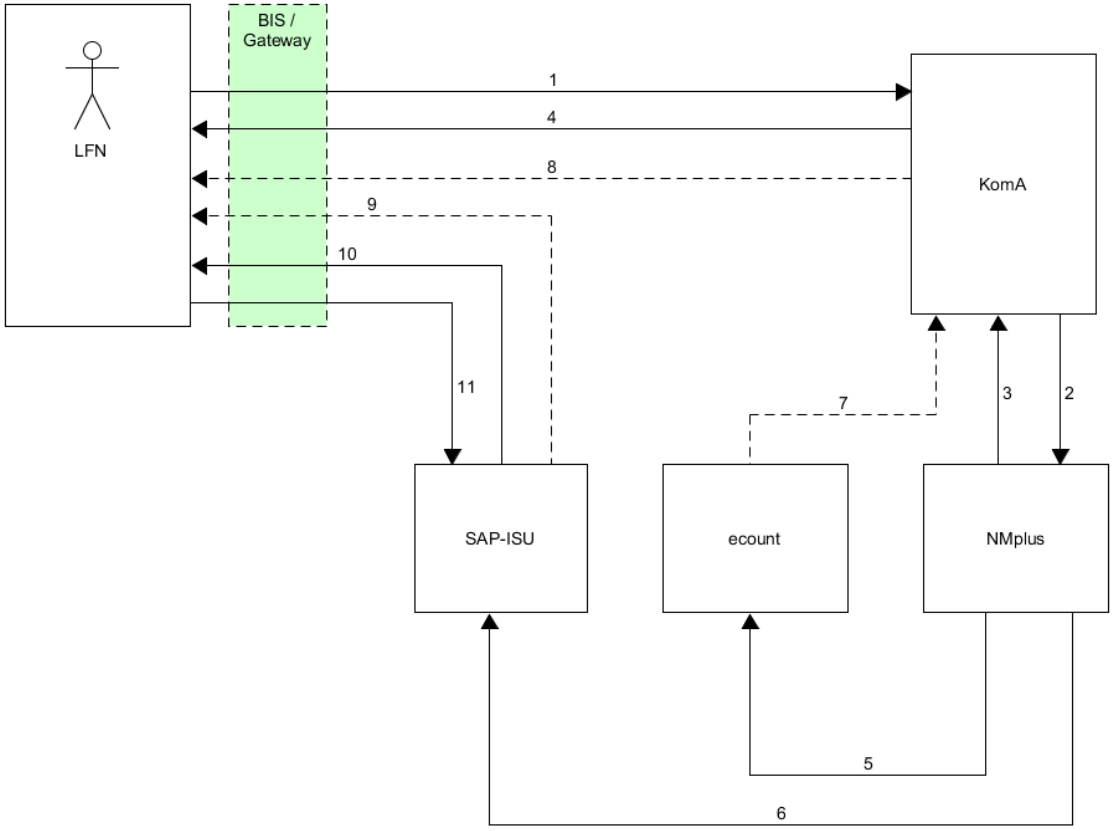

Abbildung 8: Prozesssicht der Systemlandschaft

Grundsätzlich handelt es sich um einen seriellen Prozess. Daher bilden die Beschriftungsnummern auch die Reihenfolge ab. In der Tabelle 4 sind die Einzelschritte näher spezifiziert. Außerdem werden die beteiligten EDIFACT-Subsets und Tabellen aufgelistet.

Schritt	Teilsystem	Beschreibung	Format / Tabellen
1	Lieferant Neu (LFN)	Anmeldung zur Netznutzung	UTILMD
2	KomA	Aufteilung der UTILMD-Nachricht auf Vorgänge (Meldungen)	ec_utilmd.utilmd_nachrichten ec_utilmd.utilmd_meldungen

			ec_utilmd.utm_zaehlpunkt_daten ec_utilmd.utm_zaehler_daten ec_utilmd.utm_obis_daten ec_nmp.nm_meldungen ec_nmp.nm_nachrichten
3	NMplus	Bearbeitung und Erzeugung der Zustimmung	ec_utilmd.utilmd_nachrichten ec_utilmd.utilmd_meldungen ec_utilmd.utm_zaehlpunkt_daten ec_utilmd.utm_zaehler_daten ec_utilmd.utm_obis_daten ec_nmp.nm_workflow_protokoll ec_nmp.nm_meldungen ec_nmp.nm_nachrichten ec_nmp.nm_meld_zaehler ec_nmp.idi_meld
4	KomA	Generierung des EDIFACT-Stream und Kapselung in eine EDIFACT-Datei	ec_utilmd.utilmd_nachrichten ec_utilmd.utilmd_meldungen ec_utilmd.utm_zaehlpunkt_daten ec_utilmd.utm_zaehler_daten ec_utilmd.utm_obis_daten UTILMD
5	NMplus / ecount-Wizard	Beauftragung der Bilanzierung und Lastgänge	ec_sys.vertragspartner ec_sys.vertraege ec_sys.zaehlpunkte ec_nmp.nm_lieferungen
6	NMplus	Beauftragung der Abrechnung und Übermittlung des Zählerstands an IS-U	ec_nmp.vif_nmplus_isu2 ec_nmp.vif_nmplus_rlm
7	ecount	Kommunikation von Lastgangdaten bei Leistungskunden	ec_sys.vertragspartner ec_sys.vertraege ec_sys.zaehlpunkte ec_sys.linien ec_edifact.mscons_nachrichten
8	KomA	Generierung des MSCONS-Stream und Kapselung in eine EDIFACT-Datei	ec_sys.vertragspartner ec_sys.vertraege ec_sys.zaehlpunkte ec_sys.linien ec_edifact.mscons_nachrichten

			MSCONS_LG
9	IS-U	Kommunikation von Zählerständen bei Standardlastprofil-Kunden	MSCONS_VL
10	IS-U	Erstellung einer Netznutzungsrechnung mit Bezug auf Zählerstände, Zeitraum und Zählpunkt	INVOIC
11	IS-U	Bestätigung der Netznutzungsrechnung durch LFN	REMADV

Tabelle 4: Prozesssicht der Systemlandschaft

3.3 Testobjekt robotron*ecount

Im Folgenden soll speziell auf das Testobjekt robotron*ecount bzw. NMplus eingegangen werden. Der NMplus-Workflow gliedert sich grob in die Bereiche Meldungserzeugung, Meldungsempfang, Meldungsbearbeitung, Meldungskommunikation, Bilanzierungsbeauftragung und Abrechnungsbeauftragung. In der Meldungsbearbeitung wird auf Informationen aus dem IS-U und GM zurückgegriffen. Dies erfolgt über die Schnittstellentabelle ec_nmp.idi_meld. Diese Tabelle wird einmal täglich mit aktuellen Daten aus den Teilsystemen IS-U und GM gefüllt.

Die in der Anlage Teil 3 genannten Anwendungsfälle entsprechen den genannten Workflowbereichen und sind im NMplus in ein oder mehrere Workflowstatus aufgeteilt. Die EDIFACT-Nachricht des Lieferanten wird auf die enthaltenen Geschäftsvorfälle (Meldungen) aufgegliedert und in die Datenbanktabellen geschrieben. Diese Meldungen durchlaufen dann teilautomatisch die Status (Zustände) des NMplus-Workflows. An jedem Status liegt eine PL/SQL-Prozedur vor. Diese PL/SQL-Prozeduren prüfen Bedingungen und manipulieren die Tabelleneinträge der betroffenen Meldung bzw. der Umgebungsmeldungen. Des Weiteren gibt die PL/SQL-Prozedur eine codierte Rückmeldung (p_code), die eine Statusänderung (Zustandsänderung) der betroffenen Meldung im NMplus-Workflow erwirkt. Dabei gibt es die Möglichkeit zur Änderung auf einen Ziel- oder Fehlerstatus.

Die in der Anlage Teil 3 definierten Anwendungsfälle „Pflichtfelder prüfen“ und „Pflichtfelder ergänzen“ sind Sammlungen von Workflowaktionen, welche die Meldung inhaltlich (Kann-Felder, Muss-Felder und Soll-Felder) auf Vollständigkeit und Korrektheit überprüfen. Die Regeln werden vom BDEW (Bundesverband der Energie- und Wasserwirtschaft e. V.) vorgegeben und sind in den Dokumenten UTILMD-Anwendungshandbuch [13] sowie in der Nachrichtenbeschreibung auf Basis UTILMD-Netzanschluss-Stammdaten [14] veröffentlicht. Der Anwendungsfall „Meldung identifizieren“ ordnet eine Meldung zu einer Entnahmestelle (Adresse, Zählpunkt, Zählernummer) zu. Die Regeln zur Identifizierung einer Entnahmestelle sind in der GPKE [12] S. 6 näher erläutert.

Eine Zustandsänderung des Testobjektes robotron*ecount erfolgt durch die Kombination des aktuellen Zustands des Systems, aktuellen Eingaben über die Systemschnittstellen sowie dem Zeitablauf. Der Zustand „t+1“ ist über interne oder externe Fristen definiert. Bei den externen Fristen handelt es sich zumeist um Tages- oder Wochenfristen.

Die Systemschnittstellen bestehen aus Benutzeroberflächen (GUI) und Programmierschnittstellen (API). Das Ansprechen der Programmierschnittstellen erfolgt über verschiedene Ebenen:

- PL/SQL-Prozeduren
- Parameter der EDIFACT-Meldung
- Systemparameter (Key-Mapping)

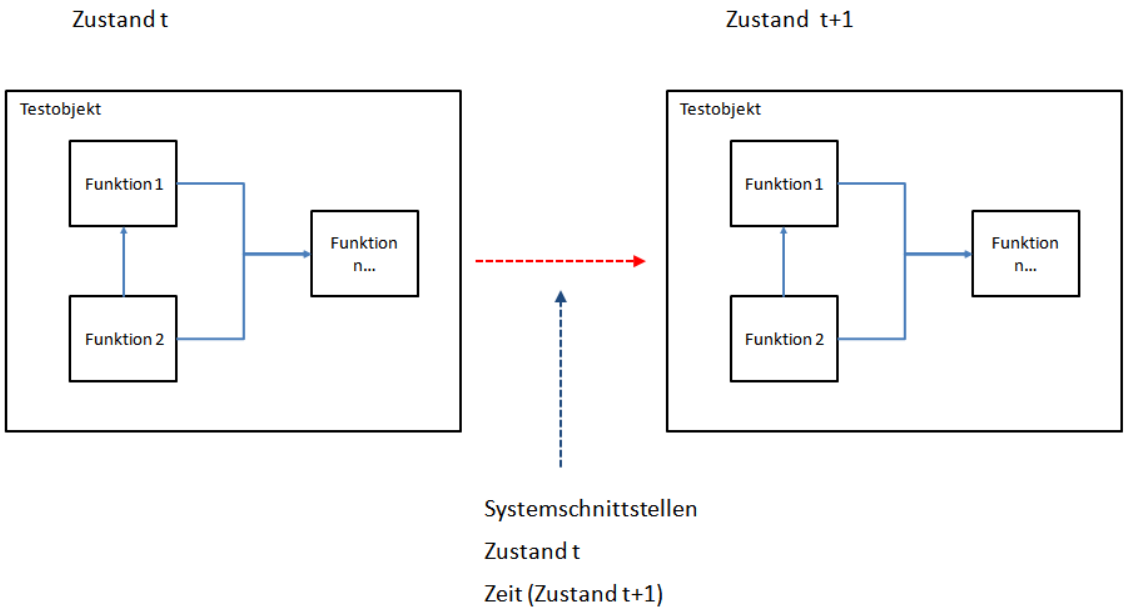

Abbildung 9: Zustandsänderungen im Testobjekt robotron*ecount

3.4 Testlandschaft

Für das Testen von Anforderungen stehen bereits Spiegelungen des produktiven Systems robotron*ecount zur Verfügung. Die Spiegelungen erfolgen in unregelmäßigen Abständen und werden als Testsysteme bezeichnet. Die Testsystemausprägungen von robotron*ecount sind über Postfächer angebunden. Damit ist die Simulation von 1:1-EDIFACT-Kommunikation möglich sowie die Analyse der vom robotron*ecount erzeugten EDIFACT-Dateien. Eine Übersicht über die entsprechenden Postfächer ist in der Tabelle 5 enthalten.

Testsystem	Eingang	Ausgang
T1	test-bis6q-edi@envia-netz.de	test-bis6-ausgang@mitnetz-strom.de
T2	test-bis6k-edi@envia-netz.de	test-bis6-ausgang@mitnetz-strom.de

Tabelle 5: EDIFACT-Schnittstellen des Testsystems

Des Weiteren besitzen die Testsysteme die gleiche Datenbank- und Tabellenstrukturen (Metadaten) wie das Produktivsystem von robotron*ecount. Der Datenbestand zwischen Testsystem und Produktivsystem kann allerdings abweichen. Der Grund liegt an den eingangs erwähnten unregelmäßigen Spiegelungen des Produktivsystems. Neben dem Testsystem stehen momentan folgende weitere Testwerkzeuge zur Verfügung:

Name	Hersteller	Beschreibung
SIGMA	ServiceNow	Anforderungskoordinierung
HP-UFT[5]	HP	Capture & Replay Tool: Oberflächentest und Schnittstellentest
EDIFACT-Generator	Westnetz	Möglichkeit zur massenhaften Erstellung von EDIFACT-Meldungen. Es fehlt Monitoring der verwendeten Testfälle und Testmetriken (Statistik)
EDIFACT Utilities	Westnetz	Einzelne EDIFACT-Dateien erstellen und analysieren
EDIFACT-Konverter	robotron	Einzelne EDIFACT-Dateien erstellen und analysieren

Tabelle 6: Testwerkzeuge

[5] Wie in der Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ [1] S. 24 festgestellt, ist das Testwerkzeug HP-UFT weiterhin nicht mit dem Testobjekt robotron*ecount kompatibel.

Eine nähere Beschreibung der oben genannten Werkzeuge sind in der Projektdokumentation „Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount“ [1] S. 19-25 enthalten.

3.5 Fazit

Wichtige Schnittstellen zum automatisierten Monitoring der Testfälle sind u. a.:

- ec_nmp.nm_nachrichten
- ec_nmp.nm_meldungen
- ec_nmp.nm_workflow_protokoll
- ec_utilmd.utilmd_nachrichten
- ec_utilmd.utilmd_meldungen
- ec_utilmd.utm_zaehlpunkt_daten
- ec_utilmd.utm_zaehler_daten
- ec_utilmd.utm_obis_daten

Die Instrumentierung der Tabellen kann z. B. über SQL-Skripte erfolgen.

Für einen vollständigen Softwaretest bzw. Regressionstest sollten folgende Teilsysteme und Schnittstellen von robotron*ecount geprüft werden:

- KomA
  - ec_utilmd.utilmd_nachrichten
  - ec_utilmd.utilmd_meldungen
  - ec_utilmd.utm_zaehlpunkt_daten
  - ec_utilmd.utm_zaehler_daten
  - ec_utilmd.utm_obis_daten
- NMplus-Workflow
  - ec_nmp.nm_meldungen
  - ec_nmp.nm_workflow_protokoll
- NMplus/ecount (Beauftragung der Bilanzierung und Lastgangdaten)
  - ec_nmp.nm_lieferungen
  - ec_sys.vertragspartner
  - ec_sys.vertraege
  - ec_sys.zaehlpunkte
  - ec_sys.linien
- Beauftragung der Abrechnung
  - ec_nmp.vif_nmplus_isu2
  - ec_nmp.vif_nmplus_rlm
- Eingehende und ausgehende EDIFACT-Dateien

Für die Erhebung von Testdaten ist die Tabelle ec_nmp.idi_meld sowie ec_nmp.nm_meldungen geeignet.

4 Lösungskonzept

In diesem Kapitel wird eine Systemstruktur beschrieben, die einen wirtschaftlichen Regressionstest für das Testobjekt robotron*ecount ermöglichen soll. Diese Systemstruktur wird im Folgenden Testautomat genannt.

Um die Wirtschaftlichkeit nachzuweisen, muss der Aufwand des Regressionstests der Aufwandsersparnis von aufgedeckten und verhinderten Softwarefehlern gegenübergestellt werden. Der Aufwand des Regressionstests ist konkret bestimmbar. Im Gegensatz dazu lässt sich der verursachte Aufwand von Softwarefehlern im Produktivsystem nur teilweise bestimmen. Der Arbeitsaufwand zur Beseitigung des Fehlers sowie Korrektur von betroffenen Prozessen ist nur durch empirische Messungen konkret ermittelbar. Trotzdem gibt es noch „weiche“, nicht direkt bestimmbare, Faktoren wie z. B. qualitative Kriterien, drohende Pönalen durch die Bundesnetzagentur und Imageschäden. Eine Auswertung diesbezüglich würde über den Rahmen dieser Arbeit hinausgehen, weshalb ein Regressionstest grundsätzlich mit möglichst geringem Aufwand abgehalten werden soll. Diese Vorgabe soll das Minimalprinzip[6] erfüllen. Somit wird durch eine Kosten-Nutzen-Analyse die Aufwandsersparnis eines automatisierten Regressionstests gegenüber einem manuellen Regressionstest betrachtet.

Momentan liegt kein Konzept für einen Regressionstest vor, weshalb zunächst die Definition eines Regressionstests erfolgen muss. Anschließend kann die quantitative Bestimmung bezüglich des Arbeitsvolumens des erstellten Regressionstests erfolgen. Als Grundlage dient die Arbeitsweise mit den bisherigen Testwerkzeugen. Darauf aufbauend kann anschließend abgeschätzt werden, inwieweit eine Testautomatisierung den Arbeitsaufwand verringert und Kosten einspart.

4.1 Regressionstest definieren

Der Testfallkatalog soll zunächst alle Prozesse bezüglich der GPKE und GeLi Gas im Testobjekt robotron*ecount abbilden. Eine Erweiterung um die Prozesse bezüglich der WiM und MPES soll nach dieser Diplomarbeit erfolgen. Daher dienen die GPKE [12], GeLi Gas [15] sowie das UTILMD Anwendungshandbuch in der Version 6.0e [13] als Grundlage des Testfallkatalogs. Des Weiteren wird in der Bearbeitung grundsätzlich nach dem Zählverfahren

[6] Das Minimalprinzip ist eine Ausprägung des ökonomischen Prinzips: „Ein vorgegebenes Ziel mit möglichst geringem Aufwand erreichen.“

der Entnahmestelle sowie nach Medium (Strom und Gas) unterschieden. Eine Vorgehensweise wäre die Erstellung von allen inhaltlich möglichen Kombinationen. Dies würde aber zu vielen, nicht sinnvollen Testfällen führen. Daher soll die Testfallerstellung auf einer empirischen Auswertung der vorliegenden produktiven Daten basieren. Das ergibt einen Testfallkatalog, welcher einen möglichst realistischen Regressionstest abbildet. Dieser Testfallkatalog unterliegt ständigen Änderungen. Der - zum Abgabetermin dieser Diplomarbeit - vorliegende Testfallkatalog liegt in der Anlage Teil 4 auf CD-ROM bei. Auf diesem Stand des Testfallkatalogs soll die Kosten-Nutzen-Analyse ansetzen. Für die Interpretation des Testfallkatalogs sind folgende Begriffsdefinitionen wichtig:

Begriff	**Definition**
Bestandsmeldung	Vorliegende Meldung im Testobjekt.
Bewegungsmeldung	In Form einer EDIFACT-Meldung. Die Bewegungsmeldung adressiert immer auf eine Bestandsmeldung im Testobjekt.
Testfall	Eine Bewegungsmeldung trifft auf eine Bestandsmeldung oder entsteht aus einer Bestandsmeldung.
Testszenario	Zusammenfassung von mehreren Testfällen.
Anwendungsfall	Kombination aus Kategorie und Transaktionsgrund (Anmeldung, Abmeldung, Stammdatenänderung, Abrechnungsbeauftragung, Geschäftsdatenanfrage, Informationsmeldung, Bilanzierungsbeauftragung und Ersatzversorgung).
Testdaten	Datenwerte der Bewegungsmeldung (z. B. Kategorie, Transaktionsgrund usw.).
Testdatentypen	Typen der Datenwerte in der Bewegungsmeldung und Bestandsmeldung (Integer,

	Unsigned Integer, Decimal, String und Set[7]).
Parameter	Eingabedaten und Ausgabedaten des Testfalls. Eingabedaten = Testdaten, Ausgabedaten = Sollergebnis.

Tabelle 7: Begriffserklärung des Testfallkatalogs

4.2 Kosten-Nutzen-Analyse

Die Kosten-Nutzen-Analyse dient später als Entscheidungsvorlage für eine evtl. Implementation der Testautomatisierung, indem die geschätzten Umsetzungskosten des Lösungskonzepts den geschätzten eingesparten Kosten durch die Automatisierung gegenübergestellt werden. Für eine fundierte Bestimmung des Testaufwands muss zunächst die Komplexitätsbestimmung der entsprechenden Testfälle erfolgen. Die Gewichtung der Testfälle ermöglicht zusätzlich eine genauere statistische Auswertung des Regressionstest-Prozesses (s. Kap. 2.6.3).

Der Komplexitätsfaktor (Formel 6) ergibt sich aus folgenden Komplexitätsmaßen:

- Formel 1: Testdatendichte (Verhältnis von Daten zu Datentypen)
- Formel 2: Testdatenkomplexität (Verhältnis von Daten zu steuernden Daten)
- Formel 3: Testfallvolumen (Verhältnis Anzahl aller Parameter des Tests zu Anzahl aller Testfälle)
- Formel 5: Testfalldichte (Verhältnis Anzahl Testfälle zu Anzahl der Anwendungsfälle)

Daher müssen folgende Werte ermittelt werden:

- Anzahl der Eingabedaten pro Testfall
- Anzahl der verwendeten Datentypen pro Testfall
- Anzahl der Parameter (Eingabe- und Ausgabedaten) aller Testfälle
- Anzahl aller Testfälle
- Anzahl aller Anwendungsfälle

Die zwei Kennzahlen Testfallvolumen und Testfalldichte beschreiben einen Durchschnittswert für den gesamten Regressionstest. Für die Bestimmung der Komplexitätsmaße Testdatendichte, Testdatenkomplexität und Testfallvolumen wurden mit dem Tool EDIFACT-

[7] Bei dem Datentyp Set handelt es sich um eine vom Anwendungshandbuch definierten Wertebereich je nach Anwendungsfall. Im Anwendungshandbuch wird dafür der Begriff „Qualifier“ verwendet.

Konverter von Robotron (s. Tabelle 6) die einzelnen EDIFACT-Dateien der Testfälle ausgewertet. Für die Bestimmung der Testfalldichte wurden Anwendungsfälle definiert. Alle vier Komplexitätsmaße sind im Testfallkatalog pro Testfall einsehbar.

Der nächste Schritt ist die Messung des manuellen zeitlichen Aufwands eines Referenztestfalls. Dafür wurde der Testfall Nr. 9 „Anmeldung LW“ mit dem Komplexitätsfaktor = 1,2 aus dem Testfallkatalog ausgewählt. Dieser Testfall wurde jeweils von vier Mitarbeitern durchgeführt. Von diesen vier Ausführungen wurde eine durchschnittliche Zeit von 36 Minuten ermittelt. Zu dieser Ausführungsdauer gehört die Vorbereitung, Durchführung, Ermittlung und Interpretation der Ergebnisse des Testfalls. Nun lässt sich aus dem vorliegenden Komplexitätsfaktor und der dazu benötigten Durchschnittszeit der Zeitaufwand (x) für den Komplexitätsfaktor = 1 ermitteln:

$$\frac{36\,min}{x} = \frac{1{,}2}{1} \quad \rightarrow \quad x = \frac{36\,min}{1{,}2} \quad \rightarrow \quad x = 30\,min$$

Die Berechnung ergibt einen durchschnittlichen manuellen Zeitaufwand von 30 Minuten für den Komplexitätsfaktor = 1. Mit dieser Grundeinheit lässt sich nun der manuelle Zeitaufwand der restlichen Testfälle ermitteln, ohne das der Zeitaufwand jedes einzelnen Testfalls separat gemessen werden muss. Der manuelle Aufwand mit dem Testautomaten liegt in der Pflege der Infrastruktur. Dies beinhaltet z. B. die manuelle Aktualisierung des Testfallkatalogs und Interpretation von ermittelten Ergebnissen. Auch bei einem automatisierten Test erhöhen komplexere Testfälle den Aufwand in der Pflege der Infrastruktur. Das Ziel der Testautomatisierung ist ein fünffach kürzerer Zeitaufwand pro Testfall. Diese Anforderung muss bei einer späteren Umsetzung des Lösungskonzepts Beachtung finden. Daher werden sechs Minuten für einen Testfall mit Komplexitätsfaktor = 1 angenommen. Nun lässt sich auch hier der anteilige Arbeitsaufwand berechnen. Eine Anpassung der Parameter nach entsprechend aktuellen Schätzungen ist möglich. Das Berechnungsmodell der Kosten-Nutzen-Analyse bleibt weiterhin bestehen.

Für die Aufwandsberechnung in der Währungseinheit Euro wurde vom Controlling ein Tagessatz eines Mitarbeiters der Vergütungsgruppe G (Sachbearbeiter) von 357,00 € ermittelt. Der Tagessatz beinhalten neben den Vergütungssätzen auch Arbeitgeberanteile und Arbeitsplatzkosten wie z. B. Büromiete, IT-Kosten, TK-Kosten und Kosten bezüglich der Personalbetreuung. Ein Arbeitstag beträgt 7,6 Stunden ohne Pausen. Das ergibt einen Kostensatz von 47,00 € pro Stunde. Die Tabelle 8 stellt die jährlichen Kosten eines manuellen und automatischen Regressionstest dar. Dazu wurde der jeweilige Gesamtaufwand der beiden Lösungen im Testfallkatalog berechnet (Spalte AA und AB). Mit dem ermittelten Stundensatz lassen sich damit die Kosten für ein Jahr bestimmen.

Lösung	Aufwand pro Monat[8] (Stunden)	Aufwand pro Jahr (Stunden)	Kosten pro Jahr (Euro)
Manueller Test	78	936	Ca. 43.992 €
Automatischer Test	16	192	Ca. 9.024 €

Tabelle 8: Aufwand- und Kostenabschätzung

Schätzungsweise könnten jährlich 34.968 € eingespart werden. Zu diesem Betrag muss aber noch eine Gegenüberstellung der Umsetzungskosten des Automatisierungskonzepts erfolgen. Zusätzlich müssen auch die Anpassungskosten des Lösungskonzepts im laufenden Betrieb mit Beachtung finden.

Das erstellte Modell zur Komplexitätsbestimmung ist allgemeingültig und kann daher auch für die Gewichtung von EDIFACT-Testfällen außerhalb dieses Lösungskonzepts Verwendung finden.

[8] Der Regressionstest soll monatlich stattfinden. Der Aufwand ist dem Testfallkatalog entnommen.

4.3 Automatisierungskonzept definieren

Die folgende Abbildung stellt ein Blockdiagramm dar, welches die grundlegenden Komponenten des Testautomats abbilden soll. Die Anforderungen an die einzelnen Komponenten werden in den folgenden Gliederungspunkten näher definiert.

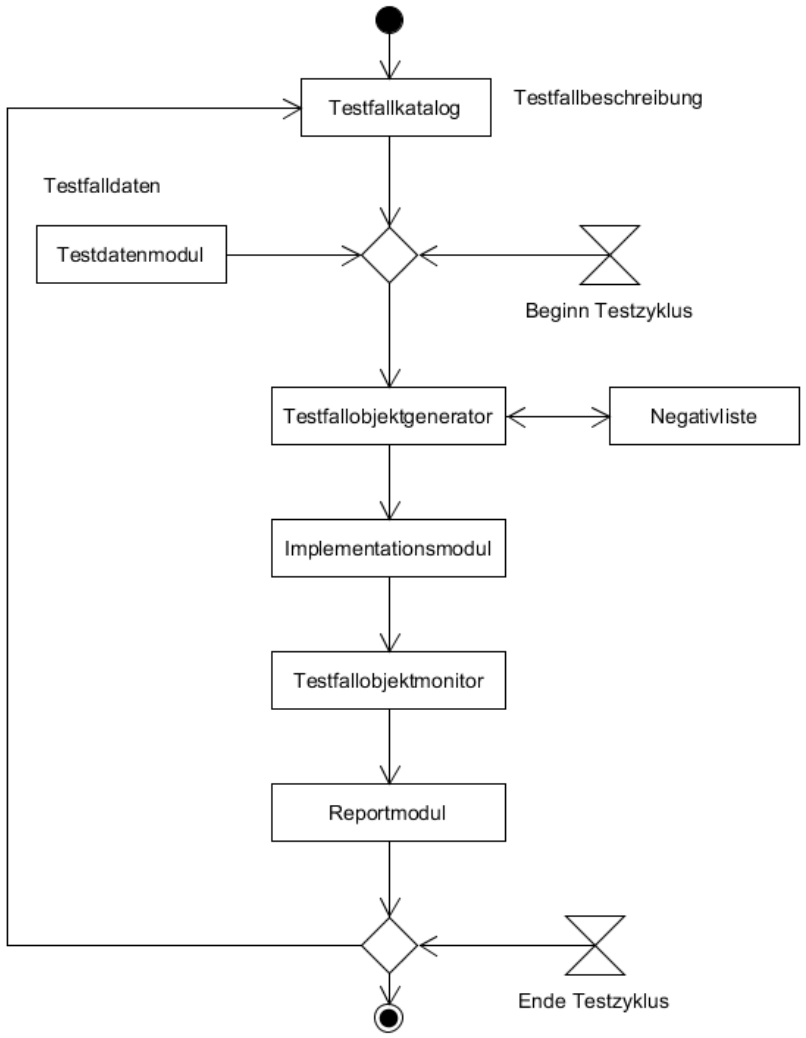

Abbildung 10: Blockdiagramm Testautomat

Der Testfallobjektgenerator erzeugt aus dem Testfallkatalog und dem Testdatenmodul Testfallobjekte. Das Testfallobjekt ist im Grunde eine Instanz der Testfallbeschreibung aus dem Testfallkatalog. Die Durchführung der Testfallobjekte übernimmt ein Implementationsmodul. Parallel erfolgt durch das Modul des Testfallobjektmonitors eine Überwachung der Testfallobjekte durch Abfragen von geeigneten Schnittstellen, die im Kapitel 3.5 näher definiert wurden. Außerdem gibt es ein Reportmodul, welches alle gesammelten Daten verdichten, aufbereiten und anzeigen soll.

Grundsätzlich können für die einzelnen Komponenten bereits bestehende Testwerkzeuge verwendet werden, sobald diese die gestellten Anforderungen erfüllen. Vor jedem Regressionstestzyklus (Zeitsignal: Testzyklus in der Abbildung 10) muss der Testfallkatalog und das Implementationsmodul aktualisiert und synchronisiert werden. Das Löschen der Negativliste darf nur erfolgen, wenn der Datenbestand des Testsystems neu aufgesetzt wird (Bereinigung). Aufgrund des großen Datenbestands von ca. 1,7 Millionen benutzbaren Bestandsmeldungen kann auf eine Bereinigung von einzelnen durchgeführten Testfällen verzichtet werden.

4.3.1 Testfallkatalog

Im Testfallkatalog ist die Testfallbeschreibung hinterlegt. Ein Testfall besteht aus folgenden Komponenten:

- Bedingungen der Bestandsmeldung
- Daten der Bewegungsmeldung (EDIFACT-Meldung)
- Atomare Handlungsanweisungen bezüglich der Durchführung
- Sollergebnis

Momentan ist noch nicht definiert, welches Capture & Replay Tool zum Einsatz kommen soll. Daher sind die Handlungsanweisungen zur Durchführung der Testfälle nicht weiter definiert. Der im Kapitel 4.1 erstellte Testfallkatalog soll in Form einer normalisierten Datenbank zum Einsatz kommen. Dazu soll folgendes Entity-Relationship-Modell (ERM) als Vorlage dienen.

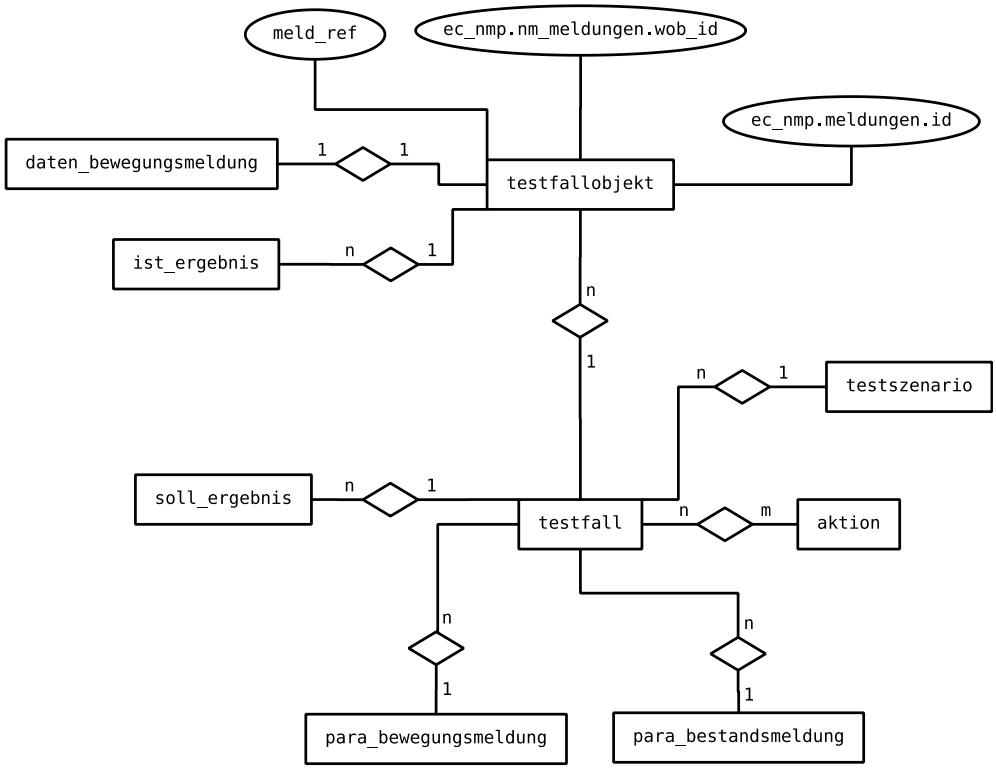

Abbildung 11: ERM des Testautomaten

Das ERM beschreibt die Struktur eines Testfalls und des daraus entstehenden Testfallobjekts als Instanz des Testfalls. Der Testfall besteht aus Bedingungen bezüglich der Bewegungsmeldung und Bestandsmeldung. Jeder Testfall besitzt dazu Sollergebnisse in der Form von Workflow- und Meldungsstatus (Kategorie, Transaktionsgrund und evtl. Antwortgrund) der Bestands- und Bewegungsmeldung. Des Weiteren gehören eine Sammlung von Handlungsanweisungen (Aktionen) zum Testfall, wie z. B. Login, diverse Workflow-Aktionen und Steuerungselemente. Nach dem Prinzip der Schlüsselwortgetriebenen Testfalldarstellung (Kapitel 2.5.2) besitzt jeder Testfall eine oder mehrere Handlungsanweisungen. Zudem kann eine Handlungsanweisung einen oder mehreren Testfällen zugeordnet werden. Damit ist die Modularität und Wiederverwendbarkeit der Handlungsanweisungen gegeben. Zusätzlich sind Testfälle zu Testszenarien kombinierbar. Die Abbildung 12 stellt die Anwendungsfälle des Testfallkatalogs dar.

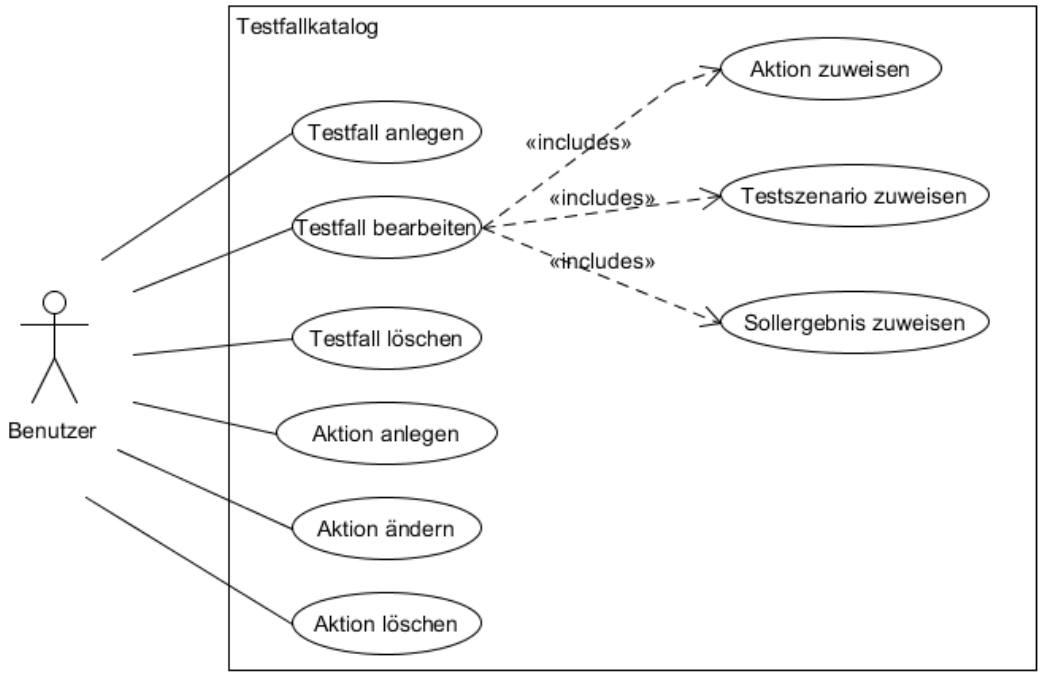

Abbildung 12: Anwendungsfalldiagramm Testfallkatalog

4.3.2 Testdatenmodul

Das Testdatenmodul stellt geeignete Testdaten bereit. Die Abfrage des Datenbestands aus dem Produktivsystem soll zu Beginn des Testrelease erfolgen. Dazu werden alle Datensätze der Tabellen ec_nmp.idi_meld, ec_nmp.idi_zaehler und ec_nmp.nm_meldungen in eine separate Tabelle (Testdatentabelle) zusammengeführt. Dieser Testdatenbestand soll im Testzyklus Bestand haben und wird dementsprechend nicht mehr geändert. Außerdem ist an der Stelle eine Negativliste vorgesehen. Sobald eine Bestandsmeldung Verwendung findet, muss diese für andere Testszenarien ausgeschlossen werden. Dies erfolgt mit einem

Listeneintrag des Zählpunkts, Primärschlüssels sowie einem Zeitstempel der betroffenen Bestandsmeldung durch die Instanz des Testobjektgenerators. Bei erneutem Aufsetzen des Testsystems kann der Inhalt der Negativliste wieder gelöscht werden. Es sollen auch einzelne Anpassungen durch den Benutzer möglich sein.

Der EDIFACT-Generator (s. Tabelle 6) beinhaltet schon Funktionen zur Erhebung von Testdaten aus dem Produktivsystem. Diese können auch für den Testautomaten Verwendung finden. Die Abbildung 13 gibt eine Übersicht über die möglichen Anwendungsfälle des Testdatenmoduls.

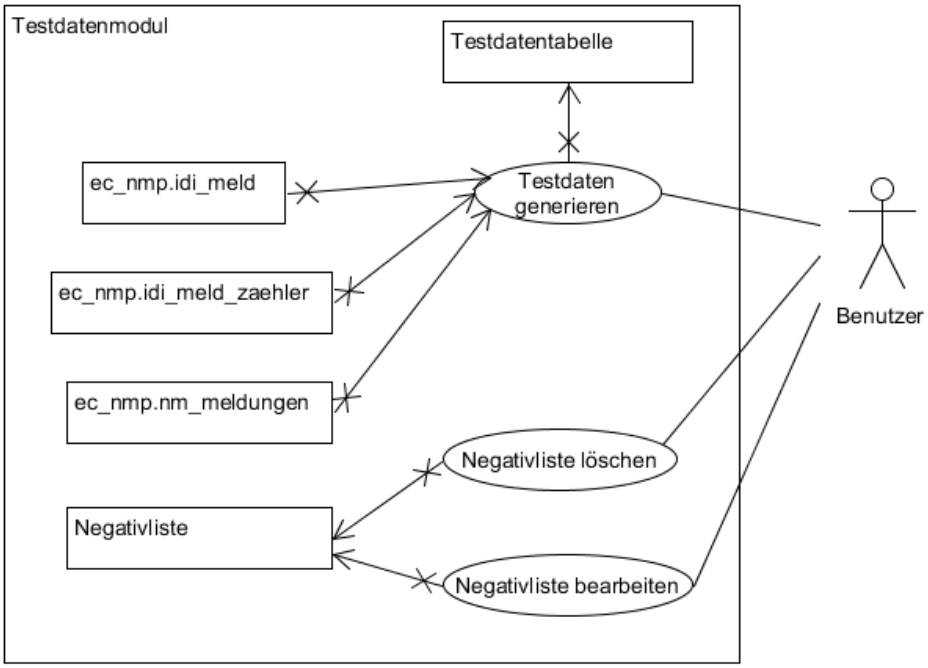

Abbildung 13: Anwendungsfalldiagramm Testdatenmodul

4.3.3 Testfallobjektgenerator

Im Testfallobjektgenerator werden die Testfallbeschreibungen mit Testdaten zusammengeführt. Dabei entsteht ein Testfallobjekt. Dieses Prinzip wurde in den Kapiteln 2.5.1 und 2.5.2 schon erläutert. Das Testfallobjekt besteht immer aus einer Bestandsmeldung, Bewegungsmeldung sowie einer Referenz auf eine EDIFACT-Meldung. Die Bestandsmeldung dient zur Adressierung im Testobjekt. Die Bewegungsmeldung beinhaltet Informationen zum Aufbau der EDIFACT-Meldung. Des Weiteren ist die Richtung und Satzart des Testfalls entscheidend. Das Attribut Satzart informiert darüber, ob eine EDIFACT-Meldung eine Original- oder Antwortmeldung ist. Das Attribut der Richtung definiert, ob eine EDIFACT-Meldung aus dem Testobjekt gesendet oder empfangen wird. Diese Unterscheidungen ergeben grundsätzliche Unterschiede in der Ausführung des Testfalls. Daher wurden die möglichen Kombinationen der Satzart und Richtung zum Begriff der Testfallart zusammengefasst. Folgende Tabelle erläutert die verschieden Testfallarten.

Nr.	Testfallart	Satzart	Richtung	Sender	Empfänger
1	Eingehende Original-meldung	O	E	EDIFACT-Generator	Testobjekt
2	Ausgehende Antwort-meldung	A	A	Testobjekt	EDIFACT-Generator und Testmonitor
3	Ausgehende Original-meldung	O	A	Testobjekt	EDIFACT-Generator und Testmonitor
4	Eingehende Antwort-meldung	A	E	EDIFACT-Generator	Testobjekt

Tabelle 9: Testfallarten

In der Abbildung 14 wird der Zusammenhang der Testfallarten nochmals bildlich dargestellt.

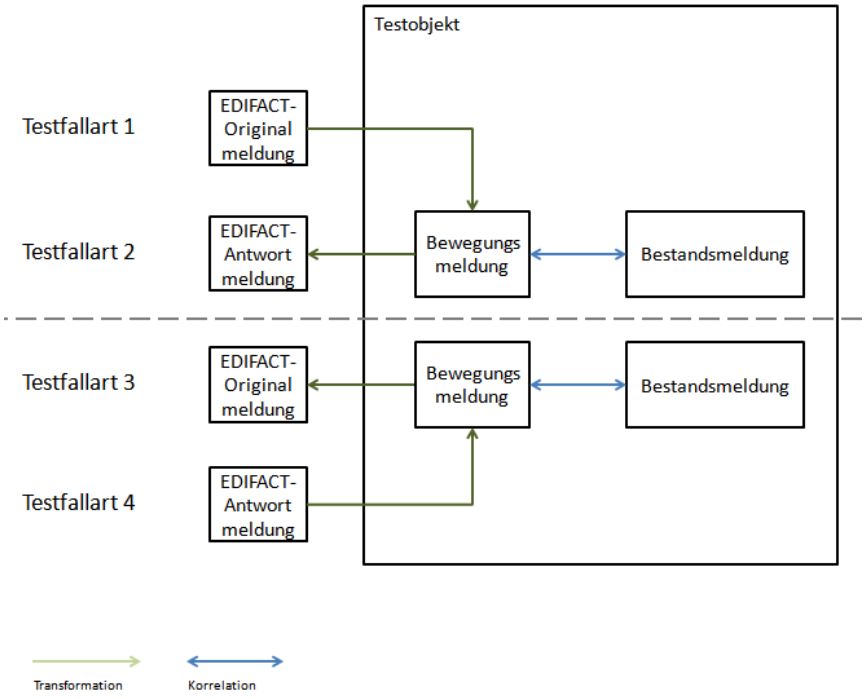

Abbildung 14: Testfallarten

Bei der Testfallart 1 wird zuerst eine EDIFACT-Meldung durch den Testfallgenerator erstellt. Diese EDIFACT-Meldung wird im Testobjekt zu einer Bewegungsmeldung transformiert. Diese trifft dann auf eine Bestandsmeldung. Die Korrelation der beiden Meldungen erzeugt eine Antwort auf die Bewegungsmeldung und damit die Testfallart 2. Bei der Testfallart 3

entsteht im Testobjekt auf Grundlage einer Bestandsmeldung eine Bewegungsmeldung. Diese Bewegungsmeldung transformiert dann zu einer EDIFACT-Meldung. Auf dieser Grundlage kann dann die Testfallart 4 ausgeführt werden.

Zur Abbildung aller Prozessausprägungen der GPKE ist eine Kombination mehrerer Testfallarten erforderlich. Die Testfallart 1 ist immer Vorrausetzung der Testfallart 2. Die Testfallart 3 ist immer die Voraussetzung der Testfallart 4. Das Testen der Abrechnungsbeauftragung und Bilanzierung erfolgt auf Grundlage der Testfallart 2 und Testfallart 3. Die Kombination von beliebig vielen Testfällen erfolgt über die Zuordnung von Testfällen zu Testszenarien. Jedes Testszenario besitzt mindestens einen Testfall. Im Testszenario ist die Reihenfolge der Testfälle festgelegt. Des Weiteren müssen Verbindungsparameter definiert sein. Dieses Prinzip soll an dem Beispiel der Prozessausprägung „Anmeldung mit Abmeldungsanfrage“ nach [12] S. 25 näher erläutert werden:

Schritt	Beschreibung	Testfallart	Verbindungsparameter
1	Anmeldung zur Netznutzung durch LFN	Eingehende Originalmeldung	-
2	Abmeldungsanfrage an LFA	Ausgehende Originalmeldung	Zählpunkt, Enddatum, LFN
2	Informationsmeldung existierende Zuordnung an LFN	Ausgehende Originalmeldung	Referenz auf Vorgangsnummer der Anmeldung
3	Zustimmung der Abmeldungsanfrage durch LFA	Eingehende Antwortmeldung	Referenz auf Vorgangsnummer der Abmeldungsanfrage
4	Mitteilung über Beendigung Zuordnung an LFA	Ausgehende Originalmeldung	Zählpunkt, Enddatum
5	Bestätigung Anmeldung an LFN	Ausgehende Antwortmeldung	Referenz auf Vorgangsnummer der Anmeldung

Tabelle 10: Testszenario "Anmeldung mit Abmeldungsanfrage"

Ein Testfallobjekt besitzt daher folgende Attribute:

- Testfallart
- Daten der Bestandsmeldung
- Daten der Bewegungsmeldung
- Referenz auf EDIFACT-Datei
- Ordnungszahl im Testszenario
- Referenz auf das Testszenario
- Handlungsanweisungen
- Ist-Ergebnisse
- Referenz auf Testfall

Zur Erstellung von EDIFACT-Dateien ist der EDIFACT-Generator der Westnetz GmbH (s. Tabelle 6) geeignet. Zusätzlich sollen die aus dem Testobjekt generierten EDIFACT-Meldungen geprüft und zur Antworterstellung verwendet werden. Zur Erfüllung der zweiten Anforderung muss eine Erweiterung des EDIFACT-Generators erfolgen.

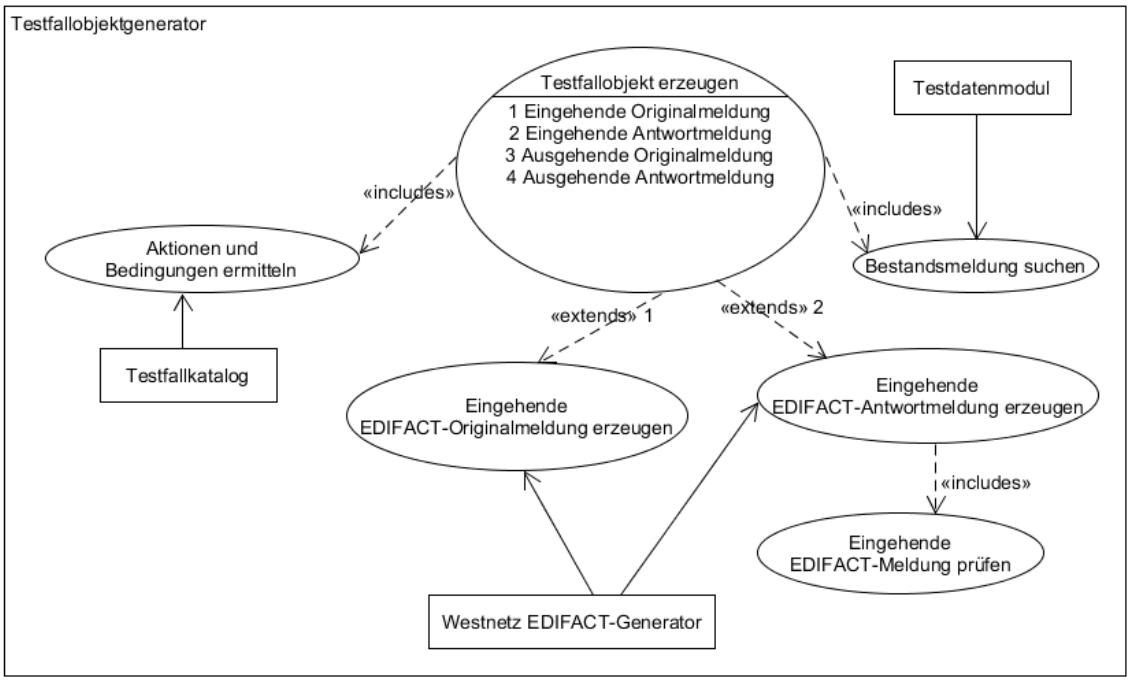

Abbildung 15: Anwendungsfalldiagramm Testfallobjektgenerator

4.3.4 Implementationsmodul

Im Implementationsmodul erfolgt die Durchführung der Testfallobjekte. Zunächst muss eine Verbindung zwischen dem Testfallobjekt und der Bewegungsmeldung im Testobjekt hergestellt werden. Die Verbindung erfolgt über den Primärschlüssel der Bewegungsmeldung. Auch hier ist eine Unterscheidung der verschiedenen Testfallarten maßgebend:

Testfallart 1 (Eingehende Originalmeldung):

Aus der Vorgangsnummer (Segment „IDE+24“ nach [14] S. 22) der vom EDIFACT-Generator erzeugten EDIFACT-Meldung kann der Primärschlüssel der Tabelle ec_nmp.nm_meldungen über die Tabelle ec_utilmd.utilmd_meldungen ermittelt werden. Damit erfolgt eine Zuordnung zwischen dem Testfallobjekt und der erzeugten EDIFACT-Meldung, die als Bewegungsmeldung das Testsystem durchläuft.

Testfallart 2 (Ausgehende Antwortmeldung):

Aus dem Primärschlüssel der Tabelle ec_nmp.nm_meldungen der zu beantwortenden Meldung kann eine Verknüpfung zur Vorgangsnummer (Segment „IDE+24“ nach [14] S. 22) der erzeugten EDIFACT-Meldung erfolgen, da nach [14] jede Antwortmeldung auf die Originalmeldung referenzieren muss.

Testfallart 3 (Ausgehende Originalmeldung):

Hier erfolgt zunächst die Generierung einer Bewegungsmeldung im Testobjekt. Die bereits vorhandene SQL-Funktion FNC_INSERT_MELD wird dabei aufgerufen und gibt den Primärschlüssel der neu erstellten Bewegungsmeldung zurück. Die Funktion verwendet dabei die RETURNING INTO Klausel von PL/SQL.

Testfallart 4 (Eingehende Antwortmeldung):

Eine durch den EDIFACT-Generator (s. Tabelle 6) erzeugte Antwortmeldung besitzt immer eine Referenz auf die Originalmeldung. Über die Vorgangsnummer (Segment „IDE+24“ nach [14] S. 22) der Originalmeldung kann über die Tabelle ec_utilmd.utilmd_meldungen der Primärschlüssel von der Tabelle ec_nmp.nm_meldungen ermittelt werden.

Die zum Testfall zugeordneten Handlungsanweisungen müssen von einem Capture & Replay-Tool interpretiert und an der Meldung des zuvor ermittelten Primärschlüssels der Tabelle ec_nmp.nm_meldungen abgearbeitet werden. Durch die eindeutige Bezeichnung der Steuerelemente im Testsystem ist eine automatisierte Durchführung der Testfälle im Testobjekt technisch möglich. Die Durchführung umfasst die Bedienung von Buttons, Textfeldern und Workflowaktionen.

4.3.5 Testfallobjektmonitor

Der Testfallobjektmonitor trägt die Ergebnisse der durchgeführten Testfälle zusammen und ordnet diese dem Testfallobjekt zu. Dementsprechend muss auch hier eine Verbindung zwischen dem Testfallobjekt und der Bewegungsmeldung vorliegen (Siehe Kapitel 4.3.4). Die erstellten Zwischenberichte ermöglichen dem Benutzer die Testüberwachung während des Testprozesses. Des Weiteren wird damit die Fehlersuche vereinfacht (Siehe Kapitel

2.5.3). Als Schnittstellen zur Verfolgung der Testfallobjekte im Testobjekt dienen die im Kapitel 3.5 ermittelten Systemtabellen. Dazu gehören u. a. die Schnittstellen ec_nmp.nm_nachrichten, ec_nmp.nm_meldungen, ec_nmp.nm_workflow_protokoll, ec_utilmd.utilmd_nachrichten und ec_utilmd.utilmd_meldungen. Diese können mit Hilfe von SQL-Skripten abgefragt werden. Die zusammengetragenen Ergebnisse des Testfallobjektmonitors bilden die Grundlage des Reportmoduls. Der Testfallobjektmonitor arbeitet parallel zum Implementationsmodul. Zur Verbesserung der Systemperformance wäre auch eine nachträgliche Abfrage der Systemtabellen möglich. Das verhindert jedoch eine Testüberwachung in Echtzeit. In der Abbildung 16 ist der Testfallobjektmonitor als Anwendungsfalldiagramm dargestellt.

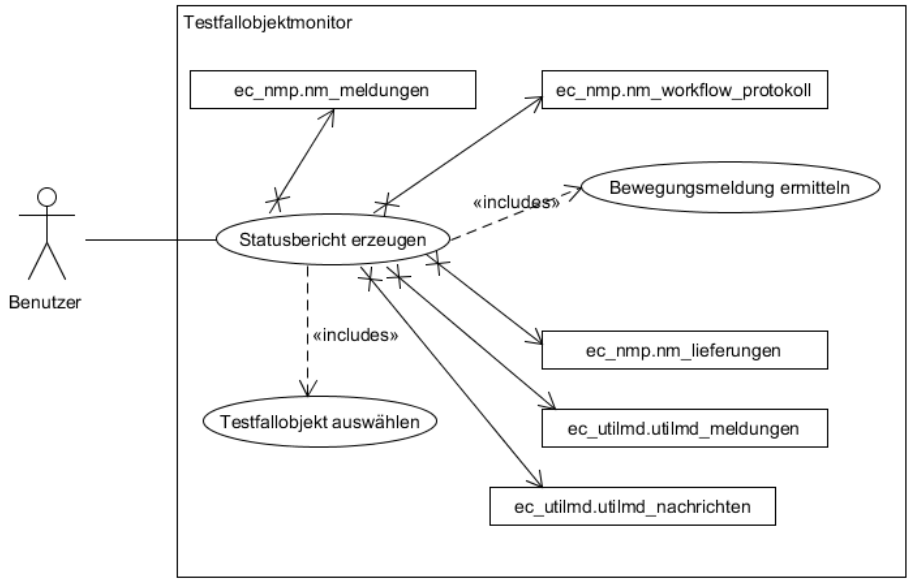

Abbildung 16: Anwendungsfalldiagramm Testfallobjektmonitor

4.3.6 Reportmodul (Metriken)

Im Reportmodul werden die vom Testfallobjektmonitor zusammengetragenen Ergebnisse aufbereitet und aggregiert. Zusätzlich soll eine bildliche Darstellung der Ergebnisse erfolgen. Dadurch soll ein Gesamtüberblick über den Testzyklus entstehen. Das Aufbereiten von Testergebnissen ist ein wichtiger Faktor für die Interpretation des Gesamtergebnisses des Testzyklus. Zusammen mit neuen Anforderungen an die Software bilden die interpretierten Testergebnisse einen hohen Einfluss auf die nötigen Veränderungen des Testfallkatalogs vor dem nächsten Testzyklus. Wichtige Metriken wären z. B.:

$$\text{Umsetzungsquote Testfälle} = \frac{\sum Komplexität\ aller\ Testfälle}{\sum Komplexität\ umgesetzter\ Testfälle}$$

Formel 7: Umsetzungsquote Testfälle

$$\text{Umsetzungsquote Testszenarien} = \frac{\sum Komplexität\ aller\ Testszenarien}{\sum Komplexität\ umgesetzter\ Testszenarien}$$

Formel 8: Umsetzungsquote Testszenarien

$$\text{Fehlerquote Testfälle} = \frac{umgesetzte\ Testfälle}{fehlerhafte\ Testfälle}$$

Formel 9: Fehlerquote Testfälle

$$\text{Fehlerquote Testszenarien} = \frac{umgesetzte\ Testszenarien}{fehlerhafte\ Testszenarien}$$

Formel 10: Fehlerquote Testszenarien

Das Reportmodul soll zusätzlich die Möglichkeit geben eigene Testmetriken zu erstellen. Damit ist eine hohe Flexibilität bezüglich der Interpretation von Testergebnissen möglich, was eine auf den jeweiligen Testzyklus angepasste Auswertung erlaubt. Die Abbildung 17 stellt die möglichen Anwendungsfälle des Reportmoduls dar.

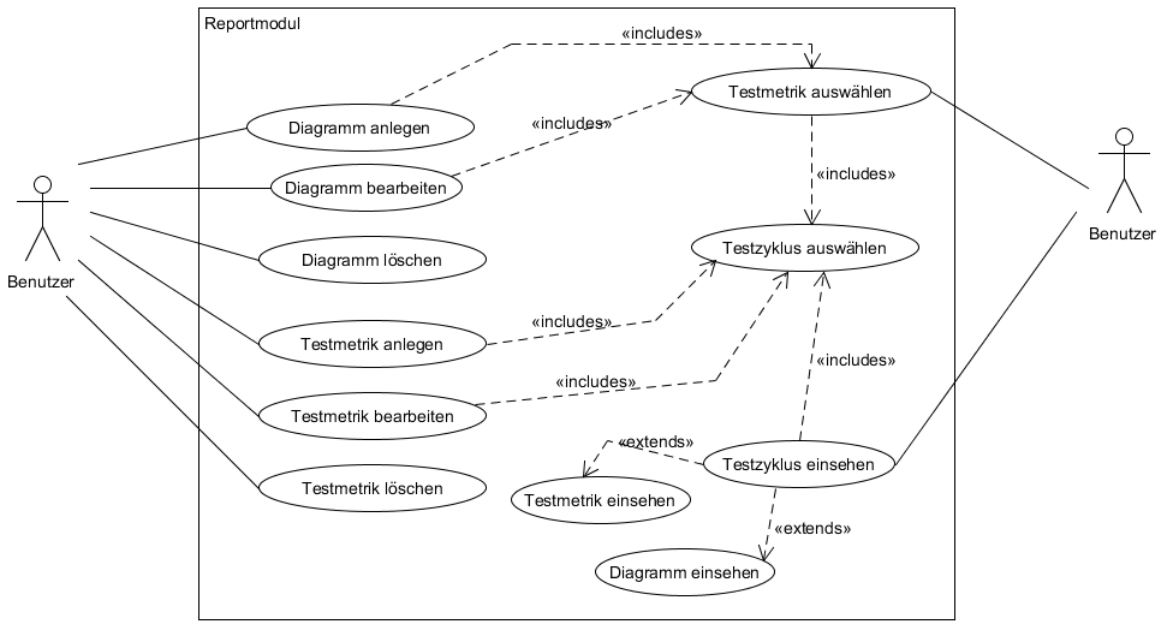

Abbildung 17: Anwendungsfalldiagramm Reportmodul

5 Zusammenfassung

Das Ziel dieser Diplomarbeit war der Entwurf eines Konzepts zur automatisierten Abwicklung von Regressionstests. Im Folgenden sollen die erreichten Ergebnisse betrachtet werden. Anschließend erfolgt ein Ausblick über die nun anstehende Entwicklung des Themas.

5.1 Fazit

Das im Kapitel 4 erstellte Lösungskonzept zur Testautomatisierung baut auf den zuvor im Kapitel 2 erörterten Konzepten und Methoden auf. Der im Punkt 2.3 besprochene Testprozess bildet die strukturelle Grundlage des Lösungskonzepts. Hier wurden schon die einzelnen Komponenten wie z. B. Testfallerstellung, Generierung von Testdaten und Testdurchführung definiert. Im Punkt 2.4 wurden technische Konzepte wie z. B. Graphische Benutzeroberflächen (GUI) und Anwendungsprogrammierschnittstellen (API) dargelegt. Diese Konzepte ermöglichen ein Zugriff auf Schnittstellen des Testobjekts und sind damit für die Implementation und Verfolgung der Testfälle von Bedeutung. Die im Punkt 2.4.3 besprochene Service-Virtualisierung stellt dagegen simulierte Schnittstellen bereit. Dieses Konzept dient zur Ersetzung von echten, nicht verfügbaren Instanzen wie z. B. Marktpartnern in den Geschäftsprozessen GPKE, GeLi Gas, WiM und MPES.

Des Weiteren wurden logische Konzepte wie die datengetriebene Testfalldarstellung und schlüsselwortgetriebene Testfalldarstellung beschrieben. Bei der datengetriebenen Testfalldarstellung erfolgt eine Trennung von Testfällen und Testdaten. Dieses Prinzip hat eine Vereinfachung der Testfälle zur Folge. Zusätzlich erhöht sich die Wiederverwendbarkeit der Testfälle. Das Prinzip der schlüsselwortgetriebenen Testfalldarstellung verfolgt einen modularen Ansatz. Hier werden Testfälle in kleinere und einfachere Teilaktivitäten aufgeteilt. Diese sollten möglichst generalisiert sein, damit auch hier eine Wiederverwendbarkeit möglich ist.

Im Kapitel 2.6.3 wurden speziell Komplexitätsmaße von Testfällen behandelt. Die Komplexitätsmaße Testdatendichte, Testdatenkomplexität, Testfallvolumen und Testfalldichte bilden den Komplexitätsfaktor eines Testfalls. Mit diesem Komplexitätsfaktor wurden die Testfälle des Testfallkatalogs quantifiziert. Dies bildet die Grundlage der Aufwandsabschätzung im Kapitel 4.2. Die dadurch ermittelte Kostenschätzung dient als Entscheidungsvorlage für eine spätere Implementation des Testautomaten. Die Wirtschaftlichkeit des Lösungskonzepts definiert sich aus zwei Aspekten. Zum einen sind das die Initialkosten bezüglich der Implementierung des Lösungskonzepts. Zum anderen sind es die wiederkehrenden Anpassungskosten im laufenden Betrieb der Testautomatisierung. Beide Parameter müssen im Verhältnis zur Aufwandsersparnis gesetzt werden. Besonders bei sich schnell ändernden

sowie heterogenen Softwarestrukturen liegt ein erhöhter Implementierungs- und Anpassungsaufwand vor.

Neben der Analyse von Konzepten und Methoden der Testautomatisierung wurde im Kapitel 3 das Testobjekt bezüglich der Softwarearchitektur mithilfe des 4+1 Sichtenmodell untersucht. Dazu gehörten u. a. Anwendungsfallsichten, logische Sichten und Prozesssichten der Architektur des Testobjekts. Durch die Analyse wurden Strukturen und Schnittstellen des Testobjekts ermittelt. Diese können z. B. für die Implementation und das Monitoring der Testfälle verwendet werden. Auch dieses Wissen ist mit in das Automatisierungskonzept eingeflossen.

5.2 Ausblick

Der nächste Schritt ist die Erstellung eines Lastenhefts aus dem Lösungskonzept. Auf dieser Grundlage kann das Umsetzungsangebot eingeholt werden. Anschließend kann ein Vergleich zwischen den Kosten des Umsetzungsangebots und der geschätzten Kostenersparnis eines automatisierten Regressionstests erfolgen. Dennoch muss ein empirischer Nachweis der Wirtschaftlichkeit des Regressionstests resultieren, indem der Aufwand des Regressionstests dem Aufwand der dadurch verhinderten Fehler gegenübergestellt wird.

Bei der Erstellung des Automatisierungskonzepts wurde das bereits vorliegende Werkzeug Westnetz EDIFACT-Generator (s. Tabelle 6) eingebunden. Bedingung ist aber eine Erweiterung des Tools um die Funktion zur Analyse von EDIFACT-Dateien. Außerdem wird für das Implementationsmodul ein geeignetes Capture & Replay Tool benötigt. Zusätzlich muss noch eine Definition der Handlungsanweisungen pro Testfall erfolgen, die das Implementationsmodul abarbeiten kann.

Das im Kapitel 4.3.6 definierte Reportmodul stellt Kennzahlen zur Analyse und Interpretation bereit. Die Interpretation der Testergebnisse sowie der dadurch abzuleitenden Schlüsse erfolgt immer noch manuell durch den Menschen. Auch hier ist eine Automation möglich. Ein Ansatz wären z. B. künstliche neuronale Netze[9] (KNN). Künstliche neuronale Netze werden mittlerweile in der Handschrift- und Spracherkennung erfolgreich eingesetzt.

[9] Mikroelektronische Schaltungen, welche funktionsmäßig den dreidimensional vernetzten Schaltungsprinzipien im Gehirn nachempfunden sind. [6] S. 568

Literatur

[1] M. Seyfarth, Vergleich von Softwarelösungen zur Testunterstützung für das Produkt robotron*ecount, Mittweida: Nicht öffentlich verfügbar, 2017.

[2] T. Bucsics, M. Baumgartner, R. Seidl und S. Gwihs, Basiswissen Testautomatisierung, 2. Auflage, Heidelberg: dpunkt.verlag, 2015.

[3] W. Czuchra, UML in logistischen Prozessen, 1. Auflage, Wiesbaden: Vieweg+Teubner Verlag | Springer Fachmedien Wiesbaden GmbH 2010, 2010.

[4] gruenderszene.de, „gruenderszene.de," 09 05 2017. [Online]. Available: https://www.gruenderszene.de/lexikon/begriffe/application-programming-interface-api. [Zugriff am 09 05 2017].

[5] D. W. Hoffmann, Software-Qualität, 2. Auflage, Spinger Vieweg, eXamen.press.

[6] P. Fischer und P. Hofer, Lexikon der Informatik, 14. Auflage, Luzern: Springer, 2008.

[7] F. Witte, Testmanagement und Softwaretest, Wiesbaden: Springer Vieweg, 2016.

[8] H. M. Sneed und S. Jungmayr, „Produkt- und Prozessmetriken für den Softwaretest," *Informatik Spektrum,* Bd. 29, Nr. 1, 2006.

[9] P. Kruchten, The 4+1 View Model of architecture, IEEE, 1995.

[10] C. Larman, Applying UML and patterns: an introduction to object-oriented analysis and design, Prentice-Hall, Inc. Upper Saddle River, 1997.

[11] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig und U. Zdun, Software-Architektur, 2. Auflage, Heidelberg: Springer/Spektrum, 2009.

[12] BK6-06-009, Geschäftsprozesse zur Kundenbelieferung mit Elektrizität, Bundesnetzagentur, 2006.

[13] BDEW, UTILMD Anwendungshandbuch, BDEW, 01.04.2016.

[14] BDEW, Nachrichtenbeschreibung auf Basis UTILMD Netzanschluss-Stammdaten (UN D.11A S3), BDEW, 01.04.2016.

[15] BK7-11-075, Geschäftsprozesse Lieferantenwechsel Gas (GeLi Gas), Bundesnetzagentur, 28.10.2011.

[16] Wikipedia, „Wikipedia; Zachman_Framework," 25 06 2017. [Online]. Available: https://de.wikipedia.org/wiki/Zachman_Framework. [Zugriff am 25 06 2017].

[17] Wikipedia, „Wikipedia; RM-ODP," 25 06 2017. [Online]. Available: https://de.wikipedia.org/wiki/RM-ODP. [Zugriff am 25 06 2017].

[18] Wikipedia, „Wikipedia," 20 05 2017. [Online]. Available: https://de.wikipedia.org/wiki/Softwaretest#Teststufen. [Zugriff am 20 05 2017].

Anlagen

Anlagen, Teil 1

Aspektübersicht der Testautomatisierung

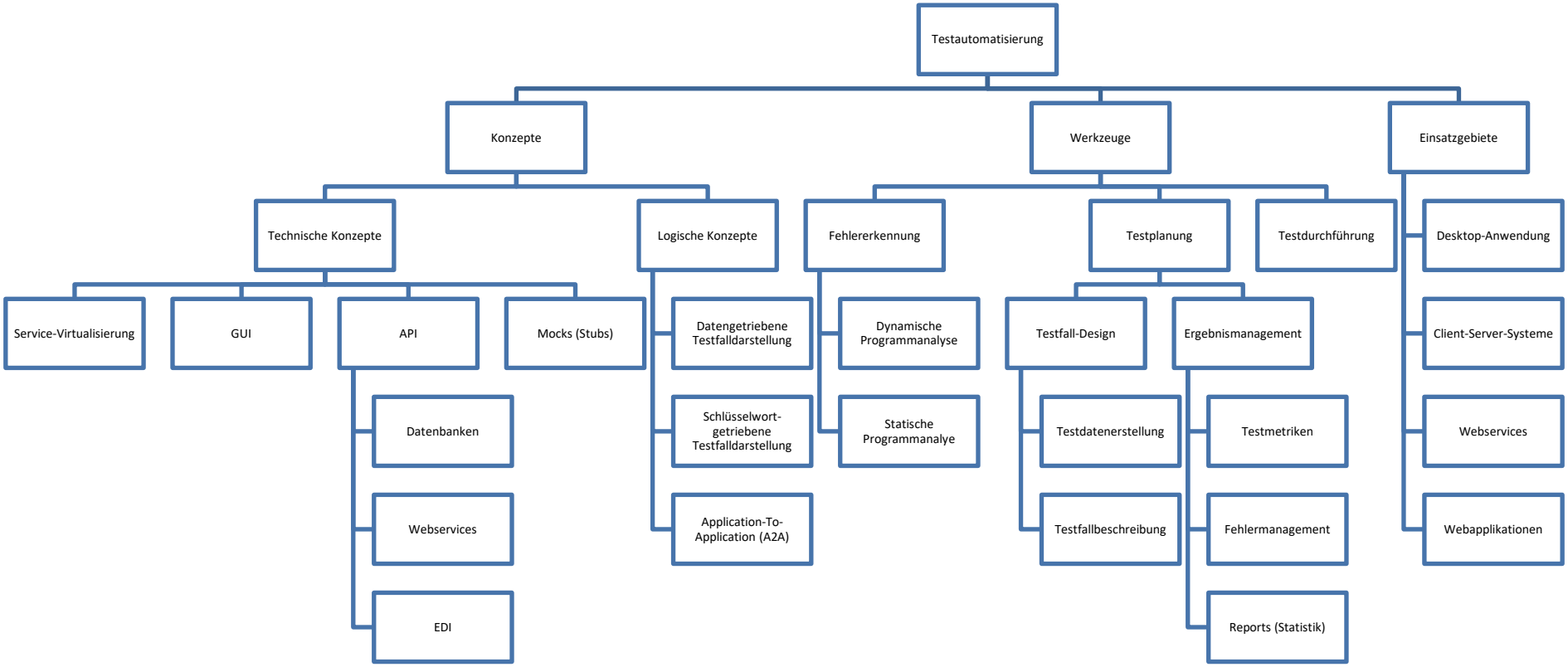

Abbildung 18: Aspektübersicht der Testautomatisierung[10]

[10] Bezugnehmend auf die Struktur des Inhaltsverzeichnis der Quelle [2].

Anlagen, Teil 2

Aufgliederung des Testprozesses

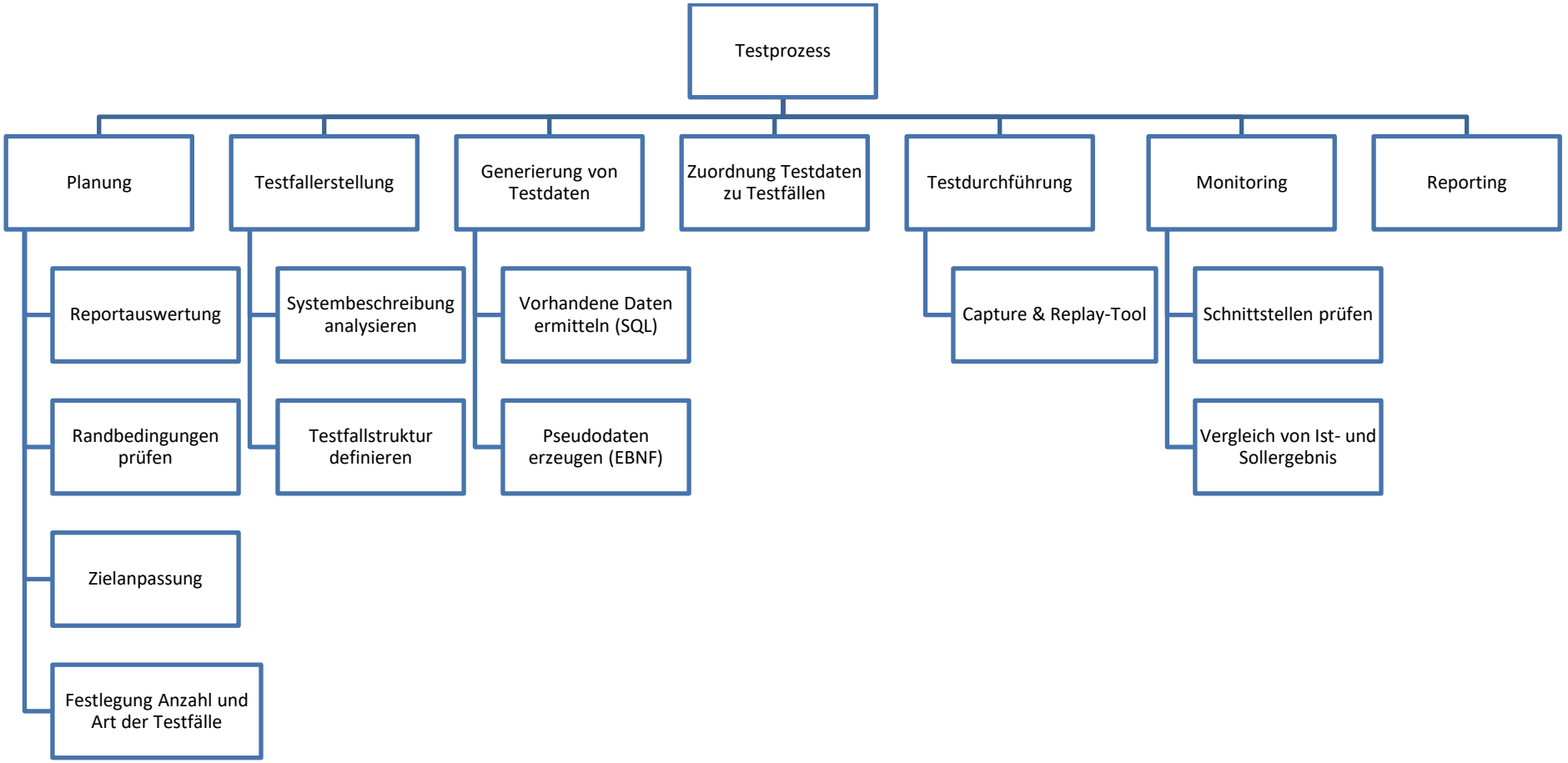

Abbildung 19: Aufgliederung des Testprozesses

Anlagen, Teil 3

Anwendungsfalldiagramm von robotron*ecount

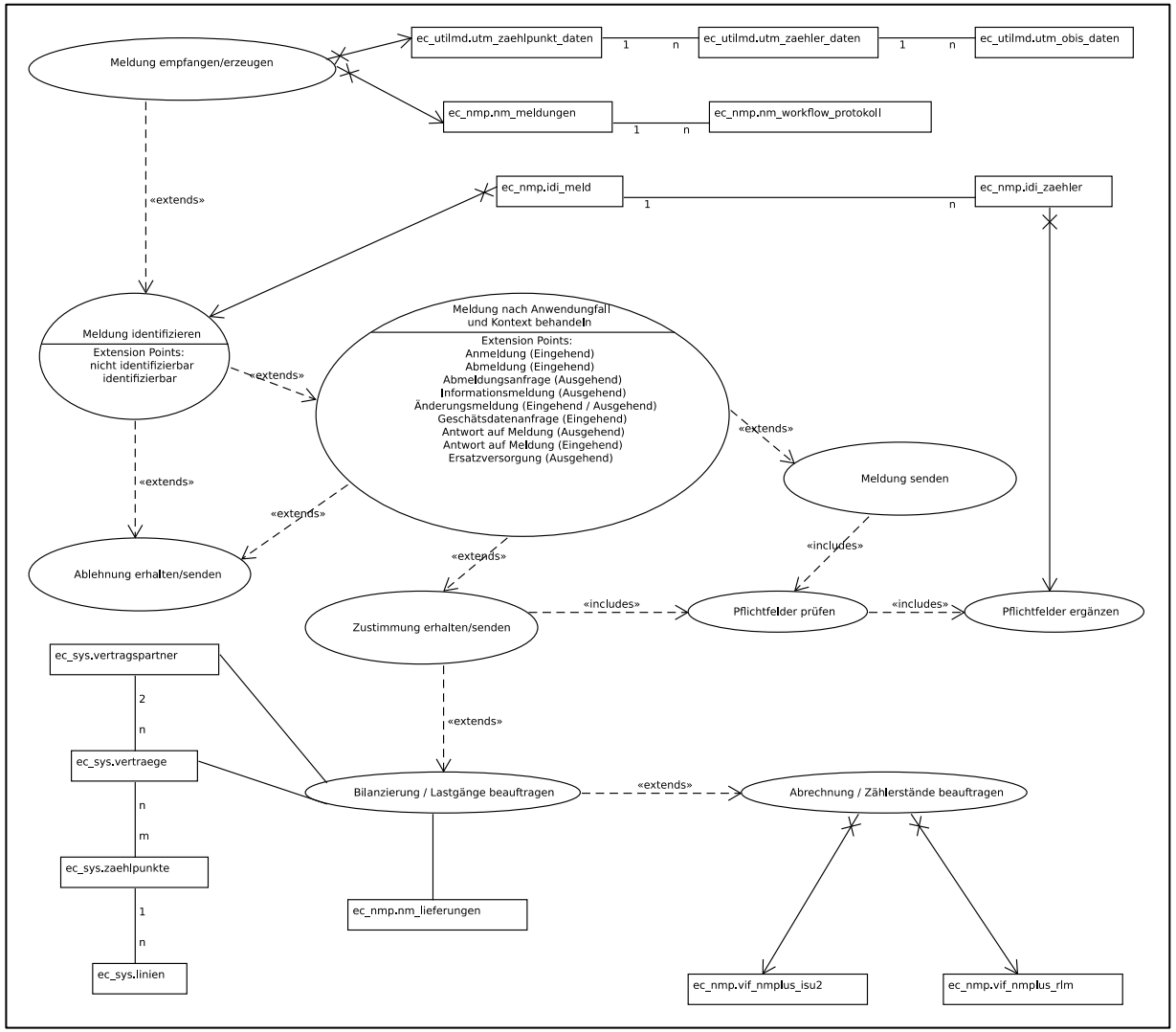

Abbildung 20: Anwendungsfalldiagramm von robotron*ecount

Anlagen, Teil 4

Testfallkatalog auf CD-ROM

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Merseburg, den 04.10.2017

Marvin Seyfarth